PATENT ABSTRACTS OF JAPAN

(11)Publication number : **07-013751**
(43)Date of publication of application : **17.01.1995**

(51)Int.CI. G06F 9/06
G06F 12/00

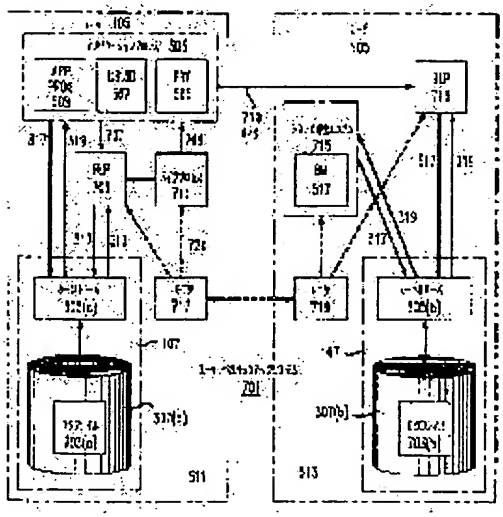
(21)Application number : **06-155409**
(22)Date of filing : **15.06.1994**

(71)Applicant : **AT & T CORP**
(72)Inventor : **FOWLER GLENN S**
HUANG YENNUN
KORN DAVID G
RAO CHUNG-HWA H

(30)Priority
Priority number : **93 80037** Priority date : **18.06.1993** Priority country : **US**

(54) SELECTIVE ADDER FOR SIDE EFFECT OF FILING SYSTEM

(57)Abstract:
PURPOSE: To provide a backup filing system(BFS) which can be used without changing an AP, OS or hardware.
CONSTITUTION: The BFS is realized by a base computer system(BCS), substitutional library (DLRL) to be dynamically linkable to the BCS, and user level process(ULP). The DLRL has the same IF as the standard DLL of a filing operation(FO). The function of the DLRL performs the same OF as the function of the DLL. Concerning this function, a message(MS) for designating the operation actually under executing is sent to the ULP of the BCS. The ULP performs the operation designated by the MS to a backup file(BF) at the BCS. The BF to be backed up is designated by identifying a sub-tree(ST) in a name space(NS) of the PFS. The ST forms a user definition NS. The processes of the BCS and BFS are monitored and a monitor process for processing a fault is used as needed so that the BFS can be made error allowable.



CLAIMS

[Claim(s)]
[Claim 1] In the equipment which adds a side effect to the actuation to the entity performed by performing the function in the library linked dynamically alternatively It substitutes for the means which matches with an entity the display of the side effect which should be added, and the library linked dynamically. It consists of an alternate library which has an alternative function and which is linked dynamically. And said alternative function It is operated by the same approach as the function in the library linked dynamically. Furthermore, alternative addition equipment of the side effect of the file system characterized by performing this side effect when the thing and the display matching means which a side effect determines whether to be what performed about an entity from a display matching means, and should be performed show.
[Claim 2] The 1st user process which performs the application program which operates about the 1st file in the 1st file system using the function from the library where the 1st is linked dynamically, The 2nd user process which operates in the 2nd file system, The library where the 2nd which has the alternative function of the function from the library where it substitutes for the library where the 1st is linked dynamically, and the 1st is linked dynamically is linked dynamically, This alternative function operates about the file in the 1st file system by the same approach as the function from the 1st library. The message which specifies the actuation to the 2nd user process is transmitted. Furthermore, [which can be performed by the 2nd user process] a means to answer this message by performing actuation about the 2nd file in the 2nd file system by which message ****** assignment is carried out -- since -- the becoming file duplicate system.

[Translation done.]

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Industrial Application] This invention relates to a computer system. Furthermore, this

invention relates to the technique used with the application programme level of a system at a detail, in order to correct the effectiveness of the call performed to the operating system.

[0002]

[Description of the Prior Art] The computer system is hierarchized. In a typical system, a hierarchy includes the hardware (for example, communication media to which a processor, memory, large capacity storage, and these components are made to link) of the lowest layer. The following layer is an operating system. An operating system controls actuation of hardware and defines a series of logical devices. The actuation in a logical device turns into actuation in the hardware controlled by the operating system. The logical device supplied by the operating system is used rather than an operating system by the program in the upper layer. These hierarchies are called the user level of a system.

[0003] The important trouble in the design of a computer system is level by which the actuation performed by the system is defined. For example, in order to operate it by an operating system's supplying a series of abundant logical devices to an application programme level, or supplying a series of small-scale fundamental logical devices, it is expected that a user's application program will combine a fundamental device if needed. Also in hardware, this is completely the same. The antinomy on the design in each level is the same.

[0004] Although a series of abundant logical devices simplify programming in the following level, the complexity of the level which supplies a logical device increases, consequently the overall flexibility of a system falls. When complicated actuation is especially performed by the lower layer of a system, these actuation must be performed without the benefit of available information by the upper layer.

[0005] The trouble on the design described here is explained to the detail into the paper it is [JIE dirty SARUZA's and others (J. H.Saltzer) "end-to-end problem in a system design" carried on ESHIEMU transactions-on computer systems (ACM Transactions on Computer Systems), the 2nd volume, No. 4 (November, 1984 issue), and 277-288 pages] entitled.

[0006] An example of an antinomy is brought about by the design of a backup file system. A backup file system gives a part of another file system or all backup. Since this is the system backed up, a backup file system also saves the known file name in [some] the system backed up instead of **** which saves a file content. The actuation performed about a file has plentifully the case where it is desirable to repeat also about the file in a backup file system.

[0007] When a backup file system exists, even if an original file system is destroyed or it becomes impossible of operation, there is no information lost. Furthermore, since a backup file system also saves a name (name) with the contents, a backup file system can be immediately used by the program currently used with the broken file system. Needless to say, although an application program can always produce the backup file system of itself, the desirable solution of a backup file system problem is always an approach that it can back up without modification of application.

[0008] Drawing 1 shows the conventional approach against the design of a backup file system. In a multiprocessing computer system, the actuation specified by the program performed in an application process is answered, and a file system performs actuation to a file. The file system itself is realized using at least one operating system process and

hardware (for example, disk drive).
[0009] The relation between an application process and an operating system process is the relation between a client and a server. A server process operates it and an application process requires that the result of the actuation should be returned to an application process. Therefore, in a system 101, the application process 103 requires that the kernel server 113 should perform file manipulation 109. For this reason, the kernel server 109 returns 111 for actuation as a result of a deed.
[0010] Needless to say, the kernel server 109 operates it by changing data with a disk drive 117. The interfaces between a disk drive 117 and the kernel server 113 are the components of the kernel server 113 called the characteristic driver 115 to the disk drive 117 of a predetermined type.
[0011] In the Prior art, the backup file system was realized in the hardware level 119 and the operating system level 107. In the system 101 of drawing 1 , it realizes in the hardware level 109. Although a disk drive 117 appears as a usual disk drive to a server 113, it includes the mirror disk 119 (a) and 119 (b).
[0012] Each file has the copy of both disks and actuation of changing a file is performed to the copy of each disk. Therefore, each disk includes the copy of a file system. Even if one side of a disk breaks, the system of other disks is still available. Needless to say, the special fault of a system 101 is needing specific hardware.
[0013] The backup file systems 121 and 123 are realized with an operating system level. A system 121 has two non-mirror disks 123 (a) and 123 (b). Each disk has a separate driver 115 (a) and separate 115 (b). When a server 113 performs file manipulation, such as creation of a file, deletion, or other modification, a driver 115 (a) makes this actuation carry out to drive 123 (a), and a driver 115 (b) makes this actuation carry out to drive 123 (b).
[0014] As a result, the ID copy of a file system exists in drive 123 (a) and drive 123 (b). Although a system 121 does not need special hardware any longer, the changed operating system is still required for it. Furthermore, since a change must be made about a driver 115 (a) and (b), this change must be made in the lowest layer of the kernel server 113.
[0015] Although the backup file system 123 also has two non-mirror disks 123 (a) and 123 (b), each disk is driven by the separate servers 125 and 127. A server 125 performs file manipulation 109 about the 1st disk 123 (a). When this file manipulation is creation of a file, deletion, or other modification, a file manipulation message also transmits a server 125 to the backup server 127. And the same file manipulation is performed after that.
[0016] Consequently, the copy of the file system currently actually used by the application process 103 exists in both disk drives 123 (a) and 123 (b). Generally the backup file system 123 is realized by the distributed computer system including many component computer systems 131. In such a system, although the application process 103, the kernel server 125, and a disk drive 123 generally exist in one component system 131 (a), the backup server 127 and a disk drive 123 (b) exist in another component system 131 (b).
[0017] What is necessary is just to change a server 125 like a system 121, so that a message may always be sent to a server 127 when changing the file about the application process 103 although a system 123 needs modification of an operating system. The backup server 127 answers this message by the completely same approach as answering

file manipulation 109 from other processes about the component system 131 (b).
[0018] Although systems 101,121 and 123 are all effective in creation of a backup file, these all have a serious defect. These systems need one of the modification of special hardware or an operating system, consequently do not become a pocket mold the 1st.
[0019] The computer system by which these systems should be performed must gain these components, when it does not have special hardware or a special operating system. Furthermore, once a system 101,121 or 123 will be in a busy condition, all the computer systems for a shift must have special hardware or a special operating system.
[0020] Since systems 101,121 and 123 operate [2nd] with a hardware level 109 or the operating system level 107, all files appear to these systems equally, and these systems only back up all the files changed according to the application process 103.
[0021] However, generally there is no need of backing up all files. When the file in a UNIX (trademark) operating system was analyzed, it became clear that 50 - 60% of the file in a file system was a temporary work file which has only the life of under a for [3 minutes]. In many cases, there is no need that almost all the things of these files back up.
[0022] However, the method of identifying these files to the kernel server 113 does not exist. It is waste of a system count resource and a system storage resource to perform needless to say and unnecessary backup.
[0023] Unnecessary backup is the special example of the aforementioned general problem. The information about this actuation is no longer used for a system, so that it will be carried out, if actuation is performed from a user. As shown by the example of backup, a result will be almost unexceptional and will be called inefficient use of a system resource.
[0024]
[Problem(s) to be Solved by the Invention] Therefore, the purpose of this invention is offering the system which can perform actuation with useful User Information without modification of an application program on user level.
[0025] Another purpose of this invention is offering the backup file system which can use the advantage of User Information about the file which can operate on user level, can carry, and should be backed up.
[0026]
[Means for Solving the Problem] The aforementioned trouble is solved by the user level backup file system of this invention. The backup file system of this invention is transposed to a series of library routines to which actuation of a series of standard library routines is emulated, and a backup file system also processes a series of standard library routines which supply a file command to a kernel server. When an application program specifies actuation of changing a file, the alternative library routine for this actuation performs this actuation.
[0027] An alternative library routine performs two actions. That is, the message which specifies the actuation which a kernel server is made to perform assignment actuation, and is actually performed by the kernel server is supplied to the user level back-end server in another computer system. Subsequently, a user level back-end server supplies the file manipulation specified by this message to the kernel server in other computer systems.
[0028] Therefore, creation and maintenance of a backup file become the side effect of the file manipulation performed by the alternative library routine. All the components of a

backup file system exist in user level, and it can be used by all the systems that can perform a routine further in an alternate library by the ability using this system, without changing hardware, an operating system, or an application program, since an alternative library routine emulates the library used from the first by the application program.
[0029] Another advantages of the user level backup file system of this invention are which file being backed up and performing the client process 303, in order to specify. This assignment is performed by duplicate tree DS. This duplicate tree DS specifies the part of an accessible file system as the client 303 which includes the file which should back up.
[0030] The code in the library routine which emulates file manipulation checks duplicate tree DS, in order to determine whether to be that by which the file at the time of a library routine performing actuation should be backed up. When it should be backed up and actuation changes [and] a file, the message which includes file manipulation is transmitted to a back-end server process.
[0031] In order to enable user level redefinition of the functionality offered by the lower layer of a computer system, generally the combination of the library routine which specifies the side effect of the actuation to a certain kind of entity, and which entity has received the bad influence according to the side effect and the DS to specify is used. For example, a user level logic file system can be defined, and this technique can be used although the map of the user level logic file system is carried out to the file system offered by the operating system.
[0032]
[Example] Hereafter, this invention is explained concretely, referring to a drawing.
[0033] Interface modification by use of a library: The computer system is hierarchized as the drawing 2 above explained. Each class forms an interface to the following upper layer. The upper layer responds for the interface of a lower layer needing, and specifies the actuation which should be performed by the lower layer. When the upper layer does not suit the interface needed by the lower layer, an adapter layer must be added between the upper layer and a lower layer.
[0034] The purpose of an adapter layer is translating the operating specification created according to the interface expected with the upper level into the operating specification for which it is needed by the interface of a lower layer. For example, an MSDOS operating system can be made to consider, if this technique is used as if it was the computer which performs a UNIX operating system for ****** PC to that user.
[0035] When an adapter layer is needed very much by many application programs, this is often realized as a series of library routines. As this name suggests, a library routine is a routine with which the user of a computer system is provided, in order that the manufacturer of the subsystem of a computer system may use it with an application program.
[0036] Drawing 2 shows the operation of the library routine for creating an adapter layer. A user program 201 has an interface 206 to the next hierarchy (they are a series of system routines in this case). However, the system routine in the computer system for which a user program 201 should be used has an interface 213. The difference between an interface 206 and an interface 213 is shown by when the configurations of a continuous line which show each interface differ in drawing 2 .
[0037] An adapter layer consists of a library routine 207. A library routine 207 has the

interface 206 for the following layers of the high order needed by the user program 201, and the interface 213 for the low-ranking following layers needed by the system routine 205. An interface actually consists of a function (function or function) call. By generating the function call needed by the interface 213, the routine in a library routine 207 answers the function call 203 needed by the interface 206, operates, and performs actuation specified by the function call 203.

[0038] As shown by the arrow 211, a system routine returns the activation result to a library routine 207, and after the system routine 215 is completed, subsequently a library routine 207 returns a result to a user program 201, as shown by the return 205.

[0039] When generating the activation code of the interface redefinition user program 201 by use of the library routine linked dynamically, the usefulness of the library routine concerning redefinition of an interface with the conventional system by the fact of having to make a library routine link to a user program 201 was low.

[0040] "Linking" here is processing which relates a call of the library routine in a user program 201 to the location of the library routine in the copy of a library routine 207. Since it had to link when an executable code was generated, the user who has only the copy of an executable code was impossible for transposing a certain library routines 207 of a series of to a series of another library routines 207.

[0041] The computer system which can make a library routine link to a user program dynamically was developed by this invention. In such a computer system, when the process which performs a user program is loaded to the memory of a computer system before activation, linking is performed.

[0042] By dynamic linking, a certain library routines of a series of can be transposed to a series of another library routines, without changing the object code of a user program, and, thereby, the behavior of the system by which a user program is operated can be changed. The explanation about dynamic linking is indicated to "the common library (Shared Libraries)" published in May, 1988 from Sun Microsystems, Inc. which does the whereabouts to Mountain View of the U.S. and California.

[0043] Drawing 3 shows how to change the behavior of a system using dynamic linking. In a system 1301, an user process 307 performs the application program 309 with which the operating system library 1315 was combined dynamically. The operating system library 1315 supplies an interface to the application program 309 shown in call 311 and return 313 list by the return 319 from the user call 317 to the kernel server 305, and the kernel server 305, in order to perform actuation specified by call 311.

[0044] In a system 2, an user process 307 performs the same application program 309, and uses the same kernel server 305. However, the operating system library 2321 is used instead of the operating system library 1315 in this case. All the things that the operating system library 1315 performs perform the operating system library 2321. However, the operating system library 2321 generates a side effect 323 further.

[0045] Therefore, it being the need in order to change it into the system 303 which also generates a side effect 323, although behavior of the system 310 is carried out like a system 301 is only making the operating system library 2321 link to a user program 309 dynamically instead of the operating system library 1315.

[0046] Creation drawing 4 of the user level name space by using the library linked dynamically shows how to control a side effect 203 using the approach of creating user level name space using the operating system library 403 linked dynamically, and this user

level name space. The entity in the computer system of a function, a file, a device, etc. is quoted in a program with a name.
[0047] Therefore, the function of the name space of a computer system is relating the name used in a program to the entity shown with this name. In the conventional computer system, the name space used by the user program is created by the operating system, and is held. In the system 401 of this invention, the operating system library 403 creates and holds one or more user level name space for user-process 409.
[0048] The one approach of using the user level name space 405 by the library routine 403 is creating the user level logic file system with which the file system with which a user program's 309 is supplied by the kernel server 305, behavior, and structure differ from these both. This logic file system can be used in order to control a side effect after that.
[0049] For example, a side effect 323 needs a system 401 a backup file system disk **** case to generate a backup file system, and the user level name space 405 specifies which file in the file system supplied by the kernel server 305 should be backed up in a backup file system. User level name space is a part of environment of an user process 409 so that clearly from drawing 4 .
[0050] A general view of a user level backup-file system: In order to form the user level backup file system which backs up automatically only the specific file of the files changed according to the application process which performs drawing 5 and the drawing 6 application program, the aforementioned library and the user level name space which were linked dynamically can be used.
[0051] Drawing 5 shows such a user level backup file system 501. A system 501 is realized by two computer systems, a main system 511 and backup system 513. A main system 511 performs the application process 503, and backup system 513 holds the backup copy of the file changed according to the application process 503. A main system 511 and backup system 513 are connected by the communication media which can transmit the message from the process performed with a main system 511 to the process performed with backup system 513.
[0052] The components of the system 501 in a main system 511 are the application process 503 and the kernel server 305 (a). The kernel server 305 (a) supplies a file system to a main system 511. In drawing 5, although a file system is shown by the local disk 307 (a) to a main system 511, this file system can also be a remote file system arranged together with another system.
[0053] In any case, the kernel server 305 (a) performs file manipulation to a file system, and call 317 is answered from the application process 503, a result 319 is returned to a process 503, and, as for a file system, the file system itself performs required actuation to a disk 307 (a). In order to perform file manipulation in the kernel server 305 (a), the application process 503 uses the library in which a dynamic link is possible.
[0054] This library is permuted in a main system 511 by the new library called lib.3d507. A library 507 answers call 317 and the file manipulation which changes a specific file also by transmitting the backup message 512 by supplying the proper call 317 to the kernel server 305 to backup system 513 is specified. The file which transmits the backup message 512 by modification is specified in the front end duplicate tree (FRT) 505.
[0055] The front end duplicate tree (FRT) 505 is held by the routine in lib.3d507, and is used so that it may be shown by the arrow 506. Therefore, the duplicate tree 505 defines

the user level logic file system which consists of a file which produces modification of the backup file in a system 513 by modification.
[0056] The component of the system in backup system 513 is the standard file system server and disk drive the back-end server 515, a user level process, the kernel server 305 (b) and a disk 307 (b), and for backup system 513. The kernel server 305 (b) supplies a file system to the back-end server 517. The data about a file system are memorized by the local disk 307 (b) in drawing 5 . However, it is also memorizable to a remote system.
[0057] The back-end server 515 performs file manipulation to the kernel server 305 (b) by call 317, and receives the result of a call from a server 305 (b). The back-end server 515 holds the back-end map 517. This back-end map 517 carries out the map of the file specified with the back-end duplicate tree 505 to the file in the file system of backup system 513 which serves as these backup. In the example which has the name space where the file system generated by the file system and the kernel server 305 (b) which were generated by the kernel server 305 (a) is the same, the back-end map 517 is unnecessary.
[0058] The approach of a system 501 of operation is shown in drawing 6 . Drawing 6 is the general-view Fig. of the gestalt of the routine 601 in the library 507 which changes a file. The routine name 603 and argument 605 which a routine has are the same as that of the name of a function and argument which are used for performing file manipulation in the library permuted by the library 507. Therefore, a call of this routine in an application program 509 calls a routine 601.
[0059] After performing all things to be prepared, a routine 601 makes the same file manipulation as the routine permuted by the routine 601 perform to the kernel server 305 (a). If actuation is successful, it will be determined whether it is shown that a routine 613 is that to which the file by which a call and the front end duplicate tree 505 are changed should back up a function 613 with the name of a file changed. When the front end duplicate tree 505 shows such, a function 615 sends the message 512 which has an argument 617 to backup system 513.
[0060] This message requires that backup system 513 should perform actuation to the completely same backup file system as the actuation performed with the file system supplied by the server 305 (a). After sending a message, the return of the routine 601 is carried out. When a file does not exist in the front end duplicate tree 505, or also when the actuation specified by the function 607 is not successful, the return of the routine 601 is carried out.
[0061] The section of the code to which the sign 611 was given in drawing 6 specifies a side effect (in this case, message 512). The description of a routine is sending a message 512 to backup system 513, only when file manipulation is successful with a main system 511. It is because it is not necessary to back up unsuccessful actuation.
[0062] Generally there are two gestalten in the file manipulation in a system 501. It is changing user level name space 405, and another is a thing for which one is realized by the front end duplicate tree 505 and the back-end map 517 and which do not change. Actuation of the 2nd gestalt is the writing to the file specified in the front end duplicate tree 505. The write-in function in lib.3d507 has the same interface as the write-in function in the library permuted by lib.3d.
[0063] In the desirable example, as an argument, this function has the integer which shows the size of the pointer and the write data-ed to the file descriptor of the integer

used by the kernel server 305 (a), and the buffer containing a write data-ed, in order to identify a file. If it requires that the write-in function in lib.3d should perform a system write-operation to the file as which the kernel server 305 (a) was specified by the file descriptor and actuation is successful, it will confirm whether the file from which the function was discriminated by the file descriptor exists in the front end duplicate tree 505, and if it exists, a function will send the write-in message 512 to the back-end server 515 in backup system 513, and will carry out a return.

[0064] A message includes information required to perform correctly the write-operation in the backup file system performed by the system write-operation in the file system which identified the file written in by the kernel server 305 (a), and was supplied by the kernel server 305 (a). If the back-end server 515 receives a message, the back-end server 515 will determine the file descriptor which uses the back-end map 517 and the kernel server 305 (b) uses for backup files, will use a system [which was supplied by the kernel server 305 (b) after that] write-in function, and will perform the write-operation to a backup file using the data and positional information which were given into the message.

[0065] The simple case of actuation of changing the user level name space 405 is file deletion. A deletion function confirms whether it is necessary to delete the information about the deleted file from the front end duplicate tree 505, and if, as for the deletion function supplied by lib.3d, it requires that the kernel server 305 (a) should delete a file first and this deletion is performed, when it is necessary to delete, a deletion function will delete this information.

[0066] Next, a deletion function sends and carries out the return of the message required for deletion to the back-end server 515. If the back-end server 515 receives this message, a deletion function will arrange a file on the back-end map 517, it will require that the kernel server 305 (b) should delete this file, and all actuation to the back-end map 517 needed for coincidence by this deletion will be performed.

[0067] A much more complicated example is renaming. Renaming of the file in the file system supplied by the kernel server 305 (a) can have the result which has the following three possibility in the user level name space 405.

[0068] ** The original name of a file is a part of user level name space 405, and when a new name is also a part of user level name space 405, a file remains in the user level name space 405.

[0069] ** The original name of a file is not a part of user level name space 405, and when a new name is not a part of user level name space 405, either, a file is added to the user level name space 405.

[0070] ** The original name of a file is a part of user level name space 405, and when a new name is not a part of user level name space 405, a file is deleted from the user level name space 405.

[0071] In the 1st case, the renaming function of lib.3d requires that the kernel server 305 (a) should rename with the file system. Subsequently, it confirms whether the renamed file exists in the user level name space 405, and if it exists, since this renaming is reflected, a renaming function will change the front end duplicate tree 505, and will require and carry out the return of the renaming [in / for a message / delivery and there] to the back-end server 515. Needless to say, this message includes an old and new path name. If the back-end server 515 receives a message, a server 515 will require renaming of the kernel server 305 (b).

[0072] In the 2nd case, a renaming function requires renaming from a server 305 (a), and the file renamed as mentioned above confirms whether exist in the user level name space 405. However, in this case, a renaming function deletes the information on a renaming file from the front end duplicate tree 505, and sends and carries out the return of the message to the back-end server 515. The message to the back-end server 515 is a deletion message of a file. Answering this message, the back-end server 515 makes the kernel server 305 (a) delete a backup file.
[0073] In the 3rd case, a renaming function requires the again above renaming. However, two kinds of messages must be sent in this case. It requires that the 1st message should create the file which has the name of a file moved to the user level name space 405 in backup system 513, the kernel server 305 (b) creates this file, and the back-end server 515 answers this message by requiring that the entry of this file should be created on the back-end map 517 after that. Then, when delivery and the back-end server 515 write the write-in message which has the current contents of the file by which the renaming function was moved to the user level name space 405 in the kernel server 305 (b) and make those contents write in the backup file in backup system 513, this write-in message is answered.
[0074] The single actuation performed by the kernel server 305 in a main system 511 (a) requires making actuation of a single string [server / 515 / back-end / server / 305 / (b) / kernel] perform so that clearly from the aforementioned explanation. Furthermore, the back-end map 517 and the front end duplicate tree 505 are in the always same condition at the time of termination of the actuation performed by the function in lib.3d507 so that clearly from the aforementioned explanation.
[0075] Implementation of a desirable example: Drawing 7 - 11 drawing 7 is the block diagram of the desirable example 701 of a user level backup file system. This desirable example is realized by the system by which a certain processor performs SunOS4.1 version of a UNIX (trademark) operating system, and other processors perform MIPS4.5 version of a UNIX (trademark) operating system. Two component groups exist in a system 701. One group performs backup file actuation and the group of another side uses a system 701 as a fault-tolerant (error permission) system. The following publication explains first the component which performs backup file actuation, and, subsequently explains the component which gives fault tolerance.
[0076] When explanation is begun from a main system 511, the application process 503 has an application program 509, library lib.3d507 (this function performs backup file actuation as a side effect of file manipulation) which can be linked dynamically, and the front end duplicate tree (FRT) 505. File manipulation is performed by the kernel server 305 (a) by the system 511. The message generated by the function in a library 507 is conveyed by backup system 513 with a pipe 710.
[0077] A pipe 710 is supplied to the application process 503 by the pipe process 711, and pipe process 711 the very thing communicates with the application process 503 with a pipe 709. The pipe process 711 supplies the single pipe 710 so that it may explain to a detail further below. A pipe 710 is used by all the application processes 503 that create backup with backup system 513.
[0078] When continuing backup system 513 in a desirable example, the back-end server 515 is divided into two processes (BLP) 716, i.e., a back-end log process, and the system call engine (SYSCALL ENG) 715. In order that both may perform file manipulation, the

kernel server 305 (b) is used. The file system maintained by the kernel server 305 other than a backup file (b) includes a log file 703 (b).
[0079] The actuation is as follows. When the engine performance of file manipulation arises by activation of the application program 509 which acquires the file identification child who specifies a pipe 710 from the pipe process 711 when the application process 503 is initialized, the function of the actuation in lib.3d507 makes the function to the file system supplied by the kernel server 305 (a) perform to the kernel server 305 (a), and transmits a message to backup system 513 through a pipe 710 further.
[0080] A message will be received by the back-end log process (BLP) 716 if a message reaches backup system 513. The back-end log process 716 carries out the log of the message to the log file 703 (b) in the file system supplied by the kernel server 305 (b). When a log file 703 (b) has a message in a file, a message is always read with the system call engine 715 in order of arrival.
[0081] In the desirable example, the back-end map 517 belongs to the system call engine 715. According to the system call engine 715 reading a message, the system call engine 715 makes the file manipulation for which the kernel server 305 (b) was asked by the message perform, and system call engine 715 the very thing responds for asking by the message, and holds the back-end map 517.
[0082] As for error permission actuation of the error permission actuation system of a system 701, an error's being detected and the detected error need to answer a system by the approach of continuing actuation. In the desirable example, detection of an error and the response to this detection are processed by WatchD (user level system which creates distributed-system error permission).
[0083] WatchD can be set to the international congress about the 23rd fault-tolerant computing held in Toulouse of the France country June 22, 1993 - 24. Software with which Hugh Anh, a wye (Huang, Y) and KINTARA, and "fault tolerance of C (Kintala, C) were realized: A technique and experience (Software Implemented Fault Tolerance:) the [Hugh Anh for whom it applied on Technologies and Experiences" and September 30, 1992, and / of a wye (Huang, Y) / United States patent application] -- a No. 07/954,549 specification (name [of invention]: -- the equipment for fault-tolerant computing --) And it is explained to the approach at the detail.
[0084] What is necessary is just to understand that a WatchD system includes one monitor process over each node of a library and a distributed system called libft in explanation of this invention. libft includes the routine which performs checkpoint actuation in which the field of the routine which specifies the field of the memory for the routine which performs actuation in which a process is registered into WatchD, and an auto backup, and these memory receives. A monitor processes the monitor user process registered also into WatchD and both monitors.
[0085] If a monitor detects that the failure occurred in the process registered into the monitor, a monitor will restart the process. When a process is restarted by the libft function, a process determines what happened. In the process which carries out the monitor of the user process to a certain node of a distributed system, a monitor moves the copy of important data (determined when this also uses a libft function) to another node of a distributed system.
[0086] When a failure occurs in the node of a monitor, the monitor of the node of another side detects this failure, and restarts the user process of the node of another side using the

present copy of important data. When a failure node is restored, the monitor restarts an user process using the critical information from the node of another side. And the message which shows that the user process was restarted is transmitted. If the monitor of the node of another side receives this message, the node of another side will end the user process performed by that node.

[0087] Generally, a WatchD monitor is arranged annularly and each monitor carries out the monitor of the monitor of the endocyclic next door. Before it becomes impossible for the number of the monitors which receive the number of endocyclic nodes and the important copy of data of an user process to restart the user process registered into WatchD, it determines how many piece failure of the node of a distributed system must be carried out.

[0088] In the desirable example, a main system 511 and backup system 513 have a WatchD monitor, respectively. The relation between a monitor and the component of a system 701 is shown by the dotted line 721. The monitor for main-system 511 is a monitor 717. As shown by the dotted line 721, a monitor 717 supervises the monitor 719 in the pipe process 711, the front end log process 705, and a system 513. A monitor 719 supervises a monitor 717, the system call engine 715, and the back-end log process 716.

[0089] As shown in drawing 7, a system 717 can deal with the failure in the front end log process 705, the pipe process 711, the system call engine 715, and the back-end log process 716, and the failure of a system 513. The design which has two parts of the system 701 which gives fault tolerance (error admissibility) has the following two main purposes.

[0090] ** About the engine performance, that the overhead of failure recovery is certainly small, ** failure, and failure recovery are certainly transparent to application, and activation application should not be stopped by any means.

[0091] The failure method of recovery is based on assumption that it is the component which WatchD can trust most within a system. This is because WatchD performs a very easy task and the self-healing of it can be carried out after failure generating.

[0092] Next, recovery from the failure of backup system 513 is explained to a detail. Recovery from the failure of other processes is also surveyed. If explanation is begun from the failure of backup system 513, in such a case, a system 701 will operate as follows. If a monitor 717 detects the failure of a system 513, the pipe process 711 is told about a monitor 717, the pipe process 711 will create the front end log process 705, and it will transpose the file descriptor of a pipe 710 to the file descriptor of the pipe 707 to the front end log process 705.

[0093] If the message function used by the application process 503 detects the failure of a pipe 710, this message function will require the new file descriptor for pipes 710 from the pipe process 711. The pipe process 711 gives the file descriptor of the pipe 707 combined with the front end log process 705 to a message function, and the message transmitted by the message function goes to the front end log process 705 instead of the back-end log process 716. If the front end log process 705 receives this message, the front end log process 705 will arrange this message to the log file 703 in a main system 511 (a).

[0094] In the desirable example, a message function detects the failure of a pipe 510 as follows. A process 503 uses a TCP-IP protocol and transmits a message to a pipe 701. In this protocol, only when a precedence message is received, the following message can be transmitted. Therefore, the message function used by the function in a library routine 507

transmits a message to a pipe 710 by transmitting two kinds of messages, i.e., the Shinsei message, and a dummy message. When a message function can transmit a dummy message, the Shinsei message reaches. If the failure of a system 513 occurs, the message transmitted through a pipe 710 does not reach and a dummy message cannot transmit it, either.

[0095] If backup system 513 is recovered, a monitor 719 will make the system call engine 715 and the back-end log process 716 restart, and will be notified to a monitor 717 after that. A monitor 717 is notified to the pipe process 711. The pipe process 711 acquires the file descriptor of a pipe 710, and terminates the front-end log process 705. If the back-end log process 716 restarts by the system 513, a process 716 will acquire the copy of a log file 703 (a) from the kernel server 305 (a), and will add this copy to a log file 703 (b). Then, the system call engine 715 resumes activation of the message in a log file 703 (b).

[0096] The message function used by lib.3d is the same approach as the approach of acquiring the file descriptor about a pipe 707, and acquires the file descriptor about a pipe 710. Next, if it tries to use the file descriptor of a pipe 710 in order that a message function may transmit a message, this attempt will go wrong and, as for a message function, a pipe file descriptor will be again required from the pipe process 711. Shortly, a message function receives the file descriptor of a pipe 710, and is again connected to a back-end log file.

[0097] The remaining fault condition is processed as follows.

** The failure monitor 717 of the pipe process 722 detects this failure. The newly restarted process searches the connection with a pipe 710 from the process condition saved by WatchD. Other processes do not notice this failure and recovery.

[0098] ** The failure monitor 719 of the system call engine 715 detects this failure, and makes the system call engine 715 restart. The newly restarted system call engine 715 is recoverable from a foreign file to the precedence checkpoint condition with the checkpoint and recovery function which are given by libft. Other processes do not notice this failure and recovery.

[0099] ** The failure monitor 719 of the back-end log process 716 detects this failure, and makes the back-end log process 716 restart. A process 716 restores the condition from a checkpoint file. A monitor 719 notifies further that the back-end log process 716 was restarted to a monitor 717, and, subsequently to the pipe process 711, notifies a monitor 717. Then, a process 711 connects a pipe 710 to the new back-end log process 716. The next store of each application file and lib.3d acquires new connection from the pipe process 711.

[0100] ** As for the failure of the failure front end log process 705 of the front end log process 705, only the failure nascent state period of a system 513 exists. If a monitor 717 detects the failure of the front end log process 705, a monitor will be notified to the pipe process 711. Subsequently, the front end log process 705 is made to restart, and a pipe 708 is re-connected to the front end log process 705. The next store of an application program 509 goes wrong, consequently the message-sending function in lib.3d acquires the file descriptor about the new pipe 708 from the pipe process 711.

[0101] implementation [of the user level name space 405]: -- drawing 8 -11 -- in order to specify all a series of files from the file system supplied by the kernel server 305 (a) to the application process 503, the user level name space 405 can be used. Drawing 8 shows

the relation between the name space 801 of the file system supplied by the kernel server 305 (a), and the user level name space 405 in the user level backup file system 701.
[0102] In name space 801, a file name is arranged in the shape of a tree. The file which forms the leaf (B, D, E, G, I, M, N) of the tree in drawing 8 contains data or a program. The remaining file is a list of other files. Such a file is called a directory. Even the name of the file which can specify all the files in name space 801 to the kernel server 305 (a) with the path name which starts by the root "/", and is specified with a path name from the root includes the name of all files. Therefore, the path name of a file "D" is /A/C/D and the path name of a file "L" is /J/K/L.
[0103] The user level backup file system 701 specifies the file which should back up by specifying the subtree of the name space 801 which includes a file. Subsequently, actuation to the file in the subtree which changes a file is performed to the backup file in backup system 513. In drawing 8 R> 8, it is chosen as what four subtrees, 803 (a), 803 (b), and 803 (c) should back up.
[0104] Therefore, modification to the data files D, E, G, I, and M or N in name space 801 will produce modification to the backup file of this data file. Similarly, modification to Directories C, F, H, and L will produce modification to these backup files. Since all the files in a subtree are backed up, the file which should back up is specified with the path name of the directory which is the root of a subtree in the user level name space 405. For example, a subtree 803 (a) is specified by a path name / A/C805 (a) in the user level name space 405.
[0105] Needless to say, the map of the user level name space 405 must be carried out also to the file system supplied with the system call engine 715. This is performed in the back-end map 517. As shown in drawing 9 , the back-end map 517 includes the entry about each open file in the user level name space 405. An entry consists of two parts, the user level name space information 903 and the backup system information 905, of 513.
[0106] The user level name space information 903 identifies the file in the user level name space 405. The backup system information 905 identifies the file in the system supplied by the kernel server 305 (b) corresponding to the file identified using user level name space information.
[0107] The back-end map 517 makes it possible to map the subtree of name space 801 in the subtree of the name space 907 of the file system which the kernel server 305 (b) supplied to the back-end log process 716 and the system call engine 715. This mapping is performed by mapping the path name of the root of the subtree of name space 801 in the path name of the root of a subtree where name space 907 corresponds.
[0108] The path name of the root is called the prefix of the path name of the file in a subtree. For example, the path name in a subtree 803 (a) has a prefix/A/C. The path name of the file E in a subtree 803 (a) is E. Subtree 909 / Z is made to correspond to a subtree 803 (a) in name space 907 by mapping a prefix/A/C in the prefix/Z of name space 907 from name space 801. After mapping is completed, modification of the file specified by the path name/A/C/E in name space 801 produces modification of the file specified by the path name/Z/E in name space 907.
[0109] detail [of the front end duplicate tree 505]: -- drawing 10 -- user level name space 405 is realized by the front end duplicate tree 505 in the desirable example. Drawing 1010 is a block diagram showing the front end duplicate tree 505 in a detail. Two main components of the front end duplicate tree 505 are RTREE1015 and the file

descriptor cache 1027. The path name of the root of the subtree 803 which has the file by which RTREE1015 should be backed up is a linked list.
[0110] The file descriptor cache 1027 is an array which relates a file descriptor to a device and an entry (inode) identifier. Therefore, the form of this implementation is as a result of the fact that the file system offered by the UNIX operating system identifies a file by the entry (inode) of the identifier of a disk at which is attained to with an integer file descriptor and a file resides by three approaches, i.e., a path name, and the file in a UNIX file system table.
[0111] The file descriptor of a file is effective only in the process which opened the file again, only while this process opens this file. Although a UNIX file system table enables the translation between a device, and inode and the present file descriptor at a path name, a device, and the list between inode(s), the direct translation between a path name and the present file descriptor is not made possible.
[0112] If it explains to a detail further succeedingly, maxtry1003 and init1005 will be used for initialization of the front end duplicate tree 505. Before an initialization function gives up maxtry1003, it shows the count tried in order to set up a pipe 710 in backup system 513. init1005 shows whether the pipe was set up or not. The RPROP array 1009 is an array of the name 1011 of the actuation which can be performed by the front end duplicate tree 505.
[0113] RTREE PTR1013 is a linked list which includes the pointer of the element of the beginning of the RTREE list 1015, i.e., a certain element of each duplicate tree 803. Each element 1017 includes the pointer 1023 to the die length 1019 of the path name 1021 of the root of the duplicate tree 803, and a path name 1021, and the next element in a linked list. The connection server 1025 is a path name in the name space 801 of the pipe 710 to backup system 513.
[0114] The FD cache 1027 is the array of the file descriptor cache entry 1029. Many entries 1029 in this array exist like an available file descriptor to the application process 503. The index of the entry of the predetermined file descriptor in the FD cache 1027 is a file descriptor. While it is shown whether an entry 1029 has an effective entry actually and being opened by the file, the application process 503 includes the status flag which shows whether a child is borne or not.
[0115] An entry 1029 also includes the identifier 1101 of the file residence device in a main system 511, and the identifier of inode of the file in a main system 511. The effective entry 1029 of each file actually opened by the subtree 803 specified by the entry in RTREE1015 exists.
[0116] The detail back-end map 517 of the back-end map 517 consists of two parts, the path name map 1113 and the opening duplicate file list 1117. The path name map 1113 only carries out the map of the path name in the name space 801 of a main system 511 to the path name of the name space 907 of backup system 513. Each entry 1115 in a map establishes the relation between the front end path name 1117 and the back-end path name 1119.
[0117] The entry which maps the root of the subtree 803 in the front end name space 907 on the root of the subtree in name space 907 is included by the path name map 1113. The back-end path name 1119 is a part of backup system information 905. In the desirable example, these mapping is specified in a system configuration file.
[0118] The opening duplicate file list 1117 includes the entry 1119 of each file which the

application process 503 opens actually in the duplicate tree 803. The user level name space information 903 in an entry 1119 includes the front end file identification child (FFID) 1105 and the front end path name (FP) 1106. The front end file identification child (FFID) 1105 consists of the device identifiers and inode identifiers of a file in a main system 511.

[0119] The front end path name (FP) 1106 is divided into the front end prefix (FPR) 1107 and the subtree path name 1108. The front end prefix (FPR) 1107 is a prefix of the subtree of the file in the front end name space 801. The subtree path name 1108 is a path name of the file in the subtree. The backup system information 905 in an entry 1117 consists of a back-end file descriptor 1111.

[0120] The back-end file descriptor 1111 is a file descriptor in the file system supplied by the kernel server 305 (b) about this file. In the desirable example, the back-end map 517 is realized as a hash table. This hash table can be accessed with both the front end file identification child 1105 and the front end path name 1106.

[0121] The approach on which the approaches of creating actuation DS 505 and 517 including DS 505 and 517 and such DS are made to act by various file manipulation is explained. In the desirable example, the application process 503 is performed with the UNIX operating system which uses a Korn shell.

[0122] A process enables it, as for a Korn shell, to set up an ENV variable. An ENV variable specifies the file performed always, when a process calls a Korn shell. The file specified by the ENV variable in the application process 503 builds the front end duplicate table 505, and since it initializes, it contains information required for the application process 503.

[0123] If created, a table 505 becomes a part of address space of the application process 503, and is available to the child process of the application process 503. This child process is created by fork system call of a UNIX operating system. Therefore, a child process inherits the parents' environment. On the other hand, an exec system call gives a new environment to a child process.

[0124] In order to create the available front end duplicate tree 505 to the child of the application process 503 created by the exec system call, lib.3d has the exec function which copies the front end duplicate tree 505 to ENV available about a new process. For this reason, even if processes are other points and it is the case where those parents' address space is not inherited, lib.3d is available to this process. Even the pipe with a name or foreign file for passing the front end duplicate tree 505 to the child process created by exec can be used for other examples.

[0125] File manipulation is explained further. Actuation of the beginning of these actuation is mounting actuation. In a UNIX operating system, mounting adds the tree of a name to the name space of an operating system. In the desirable example, the version of mounting realized by lib.3d includes the mode made to add to the user level name space 405 by using the subtree of the front end name space 801 as the duplicate tree 805.

[0126] When mounting is used in this mode, a path name argument is a path name of the root of a subtree 803 added to the user level name space 405. This function adds a subtree 803 to the user level name space 405 by adding this entry to the duplicate tree 1015 by creating the duplicate tree entry 1017 about this path name. Moreover, unmounting actuation of deleting the duplicate tree entry 1017 which has the specified path name from the duplicate tree 1015 also exists.

[0127] When the application process 503 performs opening actuation to the file in the duplicate tree 805, the opening function in lib.3d creates the file descriptor cache entry 1029 about the newly opened file, and transmits an opening message to the back-end log process 716. This opening message contains the path name in a main system 511, device identifier, and inode identifier of the opened file.
[0128] If this message is performed with the system call engine 715, an entry 901 will be created on the back-end map 517. A path name 113 is used in order to discover the file in the back-end system 513 corresponding to the file opened with the main system 511. The corresponding file descriptor of a file is arranged at the back-end file descriptor 1111.
[0129] If opened by the file, in order to identify a file, a file identification child will be used for the file manipulation in a main system 511. The message for the actuation to the backup file in backup system 513 of corresponding uses a device identifier and an inode identifier, in order to identify a file. In order to perform such a message, the system call engine 715 should just access the entry in the opening duplicate file list 1117 about the device and inode which were specified within the message. This entry includes the file descriptor 1111 required to perform backup system 513.
[0130] When the application process 503 closes the file in the duplicate tree 505, a lib.3d closing function is determined from Status field 1033 about whether the child process is using the file. When the child process is not using the file, a closing function makes an invalid the file descriptor cache entry 1029 about the file in the duplicate tree 505, and transmits the closing message containing a device identifier and an inode identifier to backup system 513.
[0131] When the system call engine 715 performs this message, in order to find the entry 1119 of this file, a device and an inode identifier are used. Subsequently, the file in backup system 513 is closed using the back-end file descriptor 1111 for identifying a file, and, finally an entry 1119 is deleted from the opening duplicate file list 1117.
[0132] As mentioned above, although an example of the formation approach of a user level backup file system and operation has been explained to a detail, it is clear to this contractor that various examples of modification of the user level backup file system of this invention are realizable. For example, it reflects that a desirable example is realized by the system which performs a UNIX operating system, and much details of a desirable example are realized by the system which performs a UNIX operating system. other operating systems -- it is, and these details are changed when realizing.
[0133] Similarly, the common library system used with the operating system of Sun is used for a desirable example. Other examples use the configuration of others of the library linked dynamically. Furthermore, the thing and functional target which were indicated by this specification can also be made to realize the DS indicated on these specifications, and a message by many of approaches of equivalent others.
[0134] Furthermore, it passes over a user level backup file system only to an example of the operation of the alternate library where it was dynamically linked for changing the behavior of a process, without changing the application program currently performed according to the process. This same technique can be used also in order to define different user level name space for processes from the name space given to the process by the operating system, and not only file backup but other services can be used for it also in order for user level name space or an operating system to add to the name of which entity of the name space given to a process.

[0135]

[Translation done.]
TECHNICAL FIELD

[Industrial Application] This invention relates to a computer system. Furthermore, this invention relates to the technique used with the application programme level of a system at a detail, in order to correct the effectiveness of the call performed to the operating system.

[Translation done.]

PRIOR ART

[Description of the Prior Art] The computer system is hierarchized. In a typical system, a hierarchy includes the hardware (for example, communication media to which a processor, memory, large capacity storage, and these components are made to link) of the lowest layer. The following layer is an operating system. An operating system controls actuation of hardware and defines a series of logical devices. The actuation in a logical device turns into actuation in the hardware controlled by the operating system. The logical device supplied by the operating system is used rather than an operating system by the program in the upper layer. These hierarchies are called the user level of a system.
[0003] The important trouble in the design of a computer system is level by which the actuation performed by the system is defined. For example, in order to operate it by an operating system's supplying a series of abundant logical devices to an application programme level, or supplying a series of small-scale fundamental logical devices, it is expected that a user's application program will combine a fundamental device if needed. Also in hardware, this is completely the same. The antinomy on the design in each level is the same.
[0004] Although a series of abundant logical devices simplify programming in the following level, the complexity of the level which supplies a logical device increases, consequently the overall flexibility of a system falls. When complicated actuation is especially performed by the lower layer of a system, these actuation must be performed without the benefit of available information by the upper layer.
[0005] The trouble on the design described here is explained to the detail into the paper it is [JIE dirty SARUZA's and others (J. H.Saltzer) "end-to-end problem in a system design" carried on ESHIEMU transactions-on computer systems (ACM Transactions on Computer Systems), the 2nd volume, No. 4 (November, 1984 issue), and 277-288 pages] entitled.
[0006] An example of an antinomy is brought about by the design of a backup file system. A backup file system gives a part of another file system or all backup. Since this is the system backed up, a backup file system also saves the known file name in [some] the system backed up instead of **** which saves a file content. The actuation performed

about a file has plentifully the case where it is desirable to repeat also about the file in a backup file system.

[0007] When a backup file system exists, even if an original file system is destroyed or it becomes impossible of operation, there is no information lost. Furthermore, since a backup file system also saves a name (name) with the contents, a backup file system can be immediately used by the program currently used with the broken file system. Needless to say, although an application program can always produce the backup file system of itself, the desirable solution of a backup file system problem is always an approach that it can back up without modification of application.

[0008] Drawing 1 shows the conventional approach against the design of a backup file system. In a multiprocessing computer system, the actuation specified by the program performed in an application process is answered, and a file system performs actuation to a file. The file system itself is realized using at least one operating system process and hardware (for example, disk drive).

[0009] The relation between an application process and an operating system process is the relation between a client and a server. A server process operates it and an application process requires that the result of the actuation should be returned to an application process. Therefore, in a system 101, the application process 103 requires that the kernel server 113 should perform file manipulation 109. For this reason, the kernel server 109 returns 111 for actuation as a result of a deed.

[0010] Needless to say, the kernel server 109 operates it by changing data with a disk drive 117. The interfaces between a disk drive 117 and the kernel server 113 are the components of the kernel server 113 called the characteristic driver 115 to the disk drive 117 of a predetermined type.

[0011] In the Prior art, the backup file system was realized in the hardware level 119 and the operating system level 107. In the system 101 of drawing 1 , it realizes in the hardware level 109. Although a disk drive 117 appears as a usual disk drive to a server 113, it includes the mirror disk 119 (a) and 119 (b).

[0012] Each file has the copy of both disks and actuation of changing a file is performed to the copy of each disk. Therefore, each disk includes the copy of a file system. Even if one side of a disk breaks, the system of other disks is still available. Needless to say, the special fault of a system 101 is needing specific hardware.

[0013] The backup file systems 121 and 123 are realized with an operating system level. A system 121 has two non-mirror disks 123 (a) and 123 (b). Each disk has a separate driver 115 (a) and separate 115 (b). When a server 113 performs file manipulation, such as creation of a file, deletion, or other modification, a driver 115 (a) makes this actuation carry out to drive 123 (a), and a driver 115 (b) makes this actuation carry out to drive 123 (b).

[0014] As a result, the ID copy of a file system exists in drive 123 (a) and drive 123 (b). Although a system 121 does not need special hardware any longer, the changed operating system is still required for it. Furthermore, since a change must be made about a driver 115 (a) and (b), this change must be made in the lowest layer of the kernel server 113.

[0015] Although the backup file system 123 also has two non-mirror disks 123 (a) and 123 (b), each disk is driven by the separate servers 125 and 127. A server 125 performs file manipulation 109 about the 1st disk 123 (a). When this file manipulation is creation of a file, deletion, or other modification, a file manipulation message also transmits a

server 125 to the backup server 127. And the same file manipulation is performed after that.
[0016] Consequently, the copy of the file system currently actually used by the application process 103 exists in both disk drives 123 (a) and 123 (b). Generally the backup file system 123 is realized by the distributed computer system including many component computer systems 131. In such a system, although the application process 103, the kernel server 125, and a disk drive 123 generally exist in one component system 131 (a), the backup server 127 and a disk drive 123 (b) exist in another component system 131 (b).
[0017] What is necessary is just to change a server 125 like a system 121, so that a message may always be sent to a server 127 when changing the file about the application process 103 although a system 123 needs modification of an operating system. The backup server 127 answers this message by the completely same approach as answering file manipulation 109 from other processes about the component system 131 (b).
[0018] Although systems 101,121 and 123 are all effective in creation of a backup file, these all have a serious defect. These systems need one of the modification of special hardware or an operating system, consequently do not become a pocket mold the 1st.
[0019] The computer system by which these systems should be performed must gain these components, when it does not have special hardware or a special operating system. Furthermore, once a system 101,121 or 123 will be in a busy condition, all the computer systems for a shift must have special hardware or a special operating system.
[0020] Since systems 101,121 and 123 operate [2nd] with a hardware level 109 or the operating system level 107, all files appear to these systems equally, and these systems only back up all the files changed according to the application process 103.
[0021] However, generally there is no need of backing up all files. When the file in a UNIX (trademark) operating system was analyzed, it became clear that 50 - 60% of the file in a file system was a temporary work file which has only the life of under a for [3 minutes]. In many cases, there is no need that almost all the things of these files back up.
[0022] However, the method of identifying these files to the kernel server 113 does not exist. It is waste of a system count resource and a system storage resource to perform needless to say and unnecessary backup.
[0023] Unnecessary backup is the special example of the aforementioned general problem. The information about this actuation is no longer used for a system, so that it will be carried out, if actuation is performed from a user. As shown by the example of backup, a result will be almost unexceptional and will be called inefficient use of a system resource.

[Translation done.]

EFFECT OF THE INVENTION

[Effect of the Invention] As explained above, according to this invention, the backup file system which can be used without changing an application program, an operating system, or hardware is obtained.

TECHNICAL PROBLEM

[Problem(s) to be Solved by the Invention] Therefore, the purpose of this invention is offering the system which can perform actuation with useful User Information without modification of an application program on user level.
[0025] Another purpose of this invention is offering the backup file system which can use the advantage of User Information about the file which can operate on user level, can carry, and should be backed up.

MEANS

[Means for Solving the Problem] The aforementioned trouble is solved by the user level backup file system of this invention. The backup file system of this invention is transposed to a series of library routines to which actuation of a series of standard library routines is emulated, and a backup file system also processes a series of standard library routines which supply a file command to a kernel server. When an application program specifies actuation of changing a file, the alternative library routine for this actuation performs this actuation.
[0027] An alternative library routine performs two actions. That is, the message which specifies the actuation which a kernel server is made to perform assignment actuation, and is actually performed by the kernel server is supplied to the user level back-end server in another computer system. Subsequently, a user level back-end server supplies the file manipulation specified by this message to the kernel server in other computer systems.
[0028] Therefore, creation and maintenance of a backup file become the side effect of the file manipulation performed by the alternative library routine. All the components of a backup file system exist in user level, and it can be used by all the systems that can perform a routine further in an alternate library by the ability using this system, without changing hardware, an operating system, or an application program, since an alternative library routine emulates the library used from the first by the application program.
[0029] Another advantages of the user level backup file system of this invention are which file being backed up and performing the client process 303, in order to specify. This assignment is performed by duplicate tree DS. This duplicate tree DS specifies the part of an accessible file system as the client 303 which includes the file which should back up.
[0030] The code in the library routine which emulates file manipulation checks duplicate tree DS, in order to determine whether to be that by which the file at the time of a library routine performing actuation should be backed up. When it should be backed up and actuation changes [and] a file, the message which includes file manipulation is transmitted to a back-end server process.
[0031] In order to enable user level redefinition of the functionality offered by the lower layer of a computer system, generally the combination of the library routine which specifies the side effect of the actuation to a certain kind of entity, and which entity has received the bad influence according to the side effect and the DS to specify is used. For example, a user level logic file system can be defined, and this technique can be used

although the map of the user level logic file system is carried out to the file system offered by the operating system.

[Translation done.]

EXAMPLE

[Example] Hereafter, this invention is explained concretely, referring to a drawing.
[0033] Interface modification by use of a library: The computer system is hierarchized as the drawing 2 above explained. Each class forms an interface to the following upper layer. The upper layer responds for the interface of a lower layer needing, and specifies the actuation which should be performed by the lower layer. When the upper layer does not suit the interface needed by the lower layer, an adapter layer must be added between the upper layer and a lower layer.
[0034] The purpose of an adapter layer is translating the operating specification created according to the interface expected with the upper level into the operating specification for which it is needed by the interface of a lower layer. For example, an MSDOS operating system can be made to consider, if this technique is used as if it was the computer which performs a UNIX operating system for ****** PC to that user.
[0035] When an adapter layer is needed very much by many application programs, this is often realized as a series of library routines. As this name suggests, a library routine is a routine with which the user of a computer system is provided, in order that the manufacturer of the subsystem of a computer system may use it with an application program.
[0036] Drawing 2 shows the operation of the library routine for creating an adapter layer. A user program 201 has an interface 206 to the next hierarchy (they are a series of system routines in this case). However, the system routine in the computer system for which a user program 201 should be used has an interface 213. The difference between an interface 206 and an interface 213 is shown by when the configurations of a continuous line which show each interface differ in drawing 2 .
[0037] An adapter layer consists of a library routine 207. A library routine 207 has the interface 206 for the following layers of the high order needed by the user program 201, and the interface 213 for the low-ranking following layers needed by the system routine 205. An interface actually consists of a function (function or function) call. By generating the function call needed by the interface 213, the routine in a library routine 207 answers the function call 203 needed by the interface 206, operates, and performs actuation specified by the function call 203.
[0038] As shown by the arrow 211, a system routine returns the activation result to a library routine 207, and after the system routine 215 is completed, subsequently a library routine 207 returns a result to a user program 201, as shown by the return 205.
[0039] When generating the activation code of the interface redefinition user program 201 by use of the library routine linked dynamically, the usefulness of the library routine concerning redefinition of an interface with the conventional system by the fact of having to make a library routine link to a user program 201 was low.

[0040] "Linking" here is processing which relates a call of the library routine in a user program 201 to the location of the library routine in the copy of a library routine 207. Since it had to link when an executable code was generated, the user who has only the copy of an executable code was impossible for transposing a certain library routines 207 of a series of to a series of another library routines 207.
[0041] The computer system which can make a library routine link to a user program dynamically was developed by this invention. In such a computer system, when the process which performs a user program is loaded to the memory of a computer system before activation, linking is performed.
[0042] By dynamic linking, a certain library routines of a series of can be transposed to a series of another library routines, without changing the object code of a user program, and, thereby, the behavior of the system by which a user program is operated can be changed. The explanation about dynamic linking is indicated to "the common library (Shared Libraries)" published in May, 1988 from Sun Microsystems, Inc. which does the whereabouts to Mountain View of the U.S. and California.
[0043] Drawing 3 shows how to change the behavior of a system using dynamic linking. In a system 1301, an user process 307 performs the application program 309 with which the operating system library 1315 was combined dynamically. The operating system library 1315 supplies an interface to the application program 309 shown in call 311 and return 313 list by the return 319 from the user call 317 to the kernel server 305, and the kernel server 305, in order to perform actuation specified by call 311.
[0044] In a system 2, an user process 307 performs the same application program 309, and uses the same kernel server 305. However, the operating system library 2321 is used instead of the operating system library 1315 in this case. All the things that the operating system library 1315 performs perform the operating system library 2321. However, the operating system library 2321 generates a side effect 323 further.
[0045] Therefore, it being the need in order to change it into the system 303 which also generates a side effect 323, although behavior of the system 310 is carried out like a system 301 is only making the operating system library 2321 link to a user program 309 dynamically instead of the operating system library 1315.
[0046] Creation drawing 4 of the user level name space by using the library linked dynamically shows how to control a side effect 203 using the approach of creating user level name space using the operating system library 403 linked dynamically, and this user level name space. The entity in the computer system of a function, a file, a device, etc. is quoted in a program with a name.
[0047] Therefore, the function of the name space of a computer system is relating the name used in a program to the entity shown with this name. In the conventional computer system, the name space used by the user program is created by the operating system, and is held. In the system 401 of this invention, the operating system library 403 creates and holds one or more user level name space for user-process 409.
[0048] The one approach of using the user level name space 405 by the library routine 403 is creating the user level logic file system with which the file system with which a user program's 309 is supplied by the kernel server 305, behavior, and structure differ from these both. This logic file system can be used in order to control a side effect after that.
[0049] For example, a side effect 323 needs a system 401 a backup file system disk ****

case to generate a backup file system, and the user level name space 405 specifies which file in the file system supplied by the kernel server 305 should be backed up in a backup file system. User level name space is a part of environment of an user process 409 so that clearly from drawing 4 .

[0050] A general view of a user level backup-file system: In order to form the user level backup file system which backs up automatically only the specific file of the files changed according to the application process which performs drawing 5 and the drawing 6 application program, the aforementioned library and the user level name space which were linked dynamically can be used.

[0051] Drawing 5 shows such a user level backup file system 501. A system 501 is realized by two computer systems, a main system 511 and backup system 513. A main system 511 performs the application process 503, and backup system 513 holds the backup copy of the file changed according to the application process 503. A main system 511 and backup system 513 are connected by the communication media which can transmit the message from the process performed with a main system 511 to the process performed with backup system 513.

[0052] The components of the system 501 in a main system 511 are the application process 503 and the kernel server 305 (a). The kernel server 305 (a) supplies a file system to a main system 511. In drawing 5, although a file system is shown by the local disk 307 (a) to a main system 511, this file system can also be a remote file system arranged together with another system.

[0053] In any case, the kernel server 305 (a) performs file manipulation to a file system, and call 317 is answered from the application process 503, a result 319 is returned to a process 503, and, as for a file system, the file system itself performs required actuation to a disk 307 (a). In order to perform file manipulation in the kernel server 305 (a), the application process 503 uses the library in which a dynamic link is possible.

[0054] This library is permuted in a main system 511 by the new library called lib.3d507. A library 507 answers call 317 and the file manipulation which changes a specific file also by transmitting the backup message 512 by supplying the proper call 317 to the kernel server 305 to backup system 513 is specified. The file which transmits the backup message 512 by modification is specified in the front end duplicate tree (FRT) 505.

[0055] The front end duplicate tree (FRT) 505 is held by the routine in lib.3d507, and is used so that it may be shown by the arrow 506. Therefore, the duplicate tree 505 defines the user level logic file system which consists of a file which produces modification of the backup file in a system 513 by modification.

[0056] The component of the system in backup system 513 is the standard file system server and disk drive the back-end server 515, a user level process, the kernel server 305 (b) and a disk 307 (b), and for backup system 513. The kernel server 305 (b) supplies a file system to the back-end server 517. The data about a file system are memorized by the local disk 307 (b) in drawing 5 . However, it is also memorizable to a remote system.

[0057] The back-end server 515 performs file manipulation to the kernel server 305 (b) by call 317, and receives the result of a call from a server 305 (b). The back-end server 515 holds the back-end map 517. This back-end map 517 carries out the map of the file specified with the back-end duplicate tree 505 to the file in the file system of backup system 513 which serves as these backup. In the example which has the name space where the file system generated by the file system and the kernel server 305 (b) which

were generated by the kernel server 305 (a) is the same, the back-end map 517 is unnecessary.
[0058] The approach of a system 501 of operation is shown in drawing 6 . Drawing 6 is the general-view Fig. of the gestalt of the routine 601 in the library 507 which changes a file. The routine name 603 and argument 605 which a routine has are the same as that of the name of a function and argument which are used for performing file manipulation in the library permuted by the library 507. Therefore, a call of this routine in an application program 509 calls a routine 601.
[0059] After performing all things to be prepared, a routine 601 makes the same file manipulation as the routine permuted by the routine 601 perform to the kernel server 305 (a). If actuation is successful, it will be determined whether it is shown that a routine 613 is that to which the file by which a call and the front end duplicate tree 505 are changed should back up a function 613 with the name of a file changed. When the front end duplicate tree 505 shows such, a function 615 sends the message 512 which has an argument 617 to backup system 513.
[0060] This message requires that backup system 513 should perform actuation to the completely same backup file system as the actuation performed with the file system supplied by the server 305 (a). After sending a message, the return of the routine 601 is carried out. When a file does not exist in the front end duplicate tree 505, or also when the actuation specified by the function 607 is not successful, the return of the routine 601 is carried out.
[0061] The section of the code to which the sign 611 was given in drawing 6 specifies a side effect (in this case, message 512). The description of a routine is sending a message 512 to backup system 513, only when file manipulation is successful with a main system 511. It is because it is not necessary to back up unsuccessful actuation.
[0062] Generally there are two gestalten in the file manipulation in a system 501. It is changing user level name space 405, and another is a thing for which one is realized by the front end duplicate tree 505 and the back-end map 517 and which do not change. Actuation of the 2nd gestalt is the writing to the file specified in the front end duplicate tree 505. The write-in function in lib.3d507 has the same interface as the write-in function in the library permuted by lib.3d.
 [0063] In the desirable example, as an argument, this function has the integer which shows the size of the pointer and the write data-ed to the file descriptor of the integer used by the kernel server 305 (a), and the buffer containing a write data-ed, in order to identify a file. If it requires that the write-in function in lib.3d should perform a system write-operation to the file as which the kernel server 305 (a) was specified by the file descriptor and actuation is successful, it will confirm whether the file from which the function was discriminated by the file descriptor exists in the front end duplicate tree 505, and if it exists, a function will send the write-in message 512 to the back-end server 515 in backup system 513, and will carry out a return.
[0064] A message includes information required to perform correctly the write-operation in the backup file system performed by the system write-operation in the file system which identified the file written in by the kernel server 305 (a), and was supplied by the kernel server 305 (a). If the back-end server 515 receives a message, the back-end server 515 will determine the file descriptor which uses the back-end map 517 and the kernel server 305 (b) uses for backup files, will use a system [which was supplied by the kernel

server 305 (b) after that] write-in function, and will perform the write-operation to a backup file using the data and positional information which were given into the message.
[0065] The simple case of actuation of changing the user level name space 405 is file deletion. A deletion function confirms whether it is necessary to delete the information about the deleted file from the front end duplicate tree 505, and if, as for the deletion function supplied by lib.3d, it requires that the kernel server 305 (a) should delete a file first and this deletion is performed, when it is necessary to delete, a deletion function will delete this information.
[0066] Next, a deletion function sends and carries out the return of the message required for deletion to the back-end server 515. If the back-end server 515 receives this message, a deletion function will arrange a file on the back-end map 517, it will require that the kernel server 305 (b) should delete this file, and all actuation to the back-end map 517 needed for coincidence by this deletion will be performed.
[0067] A much more complicated example is renaming. Renaming of the file in the file system supplied by the kernel server 305 (a) can have the result which has the following three possibility in the user level name space 405.
[0068] ** The original name of a file is a part of user level name space 405, and when a new name is also a part of user level name space 405, a file remains in the user level name space 405.
[0069] ** The original name of a file is not a part of user level name space 405, and when a new name is not a part of user level name space 405, either, a file is added to the user level name space 405.
[0070] ** The original name of a file is a part of user level name space 405, and when a new name is not a part of user level name space 405, a file is deleted from the user level name space 405.
[0071] In the 1st case, the renaming function of lib.3d requires that the kernel server 305 (a) should rename with the file system. Subsequently, it confirms whether the renamed file exists in the user level name space 405, and if it exists, since this renaming is reflected, a renaming function will change the front end duplicate tree 505, and will require and carry out the return of the renaming [in / for a message / delivery and there] to the back-end server 515. Needless to say, this message includes an old and new path name. If the back-end server 515 receives a message, a server 515 will require renaming of the kernel server 305 (b).
[0072] In the 2nd case, a renaming function requires renaming from a server 305 (a), and the file renamed as mentioned above confirms whether exist in the user level name space 405. However, in this case, a renaming function deletes the information on a renaming file from the front end duplicate tree 505, and sends and carries out the return of the message to the back-end server 515. The message to the back-end server 515 is a deletion message of a file. Answering this message, the back-end server 515 makes the kernel server 305 (a) delete a backup file.
[0073] In the 3rd case, a renaming function requires the again above renaming. However, two kinds of messages must be sent in this case. It requires that the 1st message should create the file which has the name of a file moved to the user level name space 405 in backup system 513, the kernel server 305 (b) creates this file, and the back-end server 515 answers this message by requiring that the entry of this file should be created on the back-end map 517 after that. Then, when delivery and the back-end server 515 write the

write-in message which has the current contents of the file by which the renaming function was moved to the user level name space 405 in the kernel server 305 (b) and make those contents write in the backup file in backup system 513, this write-in message is answered.

[0074] The single actuation performed by the kernel server 305 in a main system 511 (a) requires making actuation of a single string [server / 515 / back-end / server / 305 / (b) / kernel] perform so that clearly from the aforementioned explanation. Furthermore, the back-end map 517 and the front end duplicate tree 505 are in the always same condition at the time of termination of the actuation performed by the function in lib.3d507 so that clearly from the aforementioned explanation.

[0075] Implementation of a desirable example: Drawing 7 - 11 drawing 7 is the block diagram of the desirable example 701 of a user level backup file system. This desirable example is realized by the system by which a certain processor performs SunOS4.1 version of a UNIX (trademark) operating system, and other processors perform MIPS4.5 version of a UNIX (trademark) operating system. Two component groups exist in a system 701. One group performs backup file actuation and the group of another side uses a system 701 as a fault-tolerant (error permission) system. The following publication explains first the component which performs backup file actuation, and, subsequently explains the component which gives fault tolerance.

[0076] When explanation is begun from a main system 511, the application process 503 has an application program 509, library lib.3d507 (this function performs backup file actuation as a side effect of file manipulation) which can be linked dynamically, and the front end duplicate tree (FRT) 505. File manipulation is performed by the kernel server 305 (a) by the system 511. The message generated by the function in a library 507 is conveyed by backup system 513 with a pipe 710.

[0077] A pipe 710 is supplied to the application process 503 by the pipe process 711, and pipe process 711 the very thing communicates with the application process 503 with a pipe 709. The pipe process 711 supplies the single pipe 710 so that it may explain to a detail further below. A pipe 710 is used by all the application processes 503 that create backup with backup system 513.

[0078] When continuing backup system 513 in a desirable example, the back-end server 515 is divided into two processes (BLP) 716, i.e., a back-end log process, and the system call engine (SYSCALL ENG) 715. In order that both may perform file manipulation, the kernel server 305 (b) is used. The file system maintained by the kernel server 305 other than a backup file (b) includes a log file 703 (b).

[0079] The actuation is as follows. When the engine performance of file manipulation arises by activation of the application program 509 which acquires the file identification child who specifies a pipe 710 from the pipe process 711 when the application process 503 is initialized, the function of the actuation in lib.3d507 makes the function to the file system supplied by the kernel server 305 (a) perform to the kernel server 305 (a), and transmits a message to backup system 513 through a pipe 710 further.

[0080] A message will be received by the back-end log process (BLP) 716 if a message reaches backup system 513. The back-end log process 716 carries out the log of the message to the log file 703 (b) in the file system supplied by the kernel server 305 (b). When a log file 703 (b) has a message in a file, a message is always read with the system call engine 715 in order of arrival.

[0081] In the desirable example, the back-end map 517 belongs to the system call engine 715. According to the system call engine 715 reading a message, the system call engine 715 makes the file manipulation for which the kernel server 305 (b) was asked by the message perform, and system call engine 715 the very thing responds for asking by the message, and holds the back-end map 517.
[0082] As for error permission actuation of the error permission actuation system of a system 701, an error's being detected and the detected error need to answer a system by the approach of continuing actuation. In the desirable example, detection of an error and the response to this detection are processed by WatchD (user level system which creates distributed-system error permission).
[0083] WatchD can be set to the international congress about the 23rd fault-tolerant computing held in Toulouse of the France country June 22, 1993 - 24. Software with which Hugh Anh, a wye (Huang, Y) and KINTARA, and "fault tolerance of C (Kintala, C) were realized: A technique and experience (Software Implemented Fault Tolerance:) the [Hugh Anh for whom it applied on Technologies and Experiences" and September 30, 1992, and / of a wye (Huang, Y) / United States patent application] -- a No. 07/954,549 specification (name [of invention]: -- the equipment for fault-tolerant computing --) And it is explained to the approach at the detail.
[0084] What is necessary is just to understand that a WatchD system includes one monitor process over each node of a library and a distributed system called libft in explanation of this invention. libft includes the routine which performs checkpoint actuation in which the field of the routine which specifies the field of the memory for the routine which performs actuation in which a process is registered into WatchD, and an auto backup, and these memory receives. A monitor processes the monitor user process registered also into WatchD and both monitors.
[0085] If a monitor detects that the failure occurred in the process registered into the monitor, a monitor will restart the process. When a process is restarted by the libft function, a process determines what happened. In the process which carries out the monitor of the user process to a certain node of a distributed system, a monitor moves the copy of important data (determined when this also uses a libft function) to another node of a distributed system.
[0086] When a failure occurs in the node of a monitor, the monitor of the node of another side detects this failure, and restarts the user process of the node of another side using the present copy of important data. When a failure node is restored, the monitor restarts an user process using the critical information from the node of another side. And the message which shows that the user process was restarted is transmitted. If the monitor of the node of another side receives this message, the node of another side will end the user process performed by that node.
[0087] Generally, a WatchD monitor is arranged annularly and each monitor carries out the monitor of the monitor of the endocyclic next door. Before it becomes impossible for the number of the monitors which receive the number of endocyclic nodes and the important copy of data of an user process to restart the user process registered into WatchD, it determines how many piece failure of the node of a distributed system must be carried out.
[0088] In the desirable example, a main system 511 and backup system 513 have a WatchD monitor, respectively. The relation between a monitor and the component of a

system 701 is shown by the dotted line 721. The monitor for main-system 511 is a monitor 717. As shown by the dotted line 721, a monitor 717 supervises the monitor 719 in the pipe process 711, the front end log process 705, and a system 513. A monitor 719 supervises a monitor 717, the system call engine 715, and the back-end log process 716.
[0089] As shown in drawing 7, a system 717 can deal with the failure in the front end log process 705, the pipe process 711, the system call engine 715, and the back-end log process 716, and the failure of a system 513. The design which has two parts of the system 701 which gives fault tolerance (error admissibility) has the following two main purposes.
[0090] ** About the engine performance, that the overhead of failure recovery is certainly small, ** failure, and failure recovery are certainly transparent to application, and activation application should not be stopped by any means.
[0091] The failure method of recovery is based on assumption that it is the component which WatchD can trust most within a system. This is because WatchD performs a very easy task and the self-healing of it can be carried out after failure generating.
[0092] Next, recovery from the failure of backup system 513 is explained to a detail. Recovery from the failure of other processes is also surveyed. If explanation is begun from the failure of backup system 513, in such a case, a system 701 will operate as follows. If a monitor 717 detects the failure of a system 513, the pipe process 711 is told about a monitor 717, the pipe process 711 will create the front end log process 705, and it will transpose the file descriptor of a pipe 710 to the file descriptor of the pipe 707 to the front end log process 705.
[0093] If the message function used by the application process 503 detects the failure of a pipe 710, this message function will require the new file descriptor for pipes 710 from the pipe process 711. The pipe process 711 gives the file descriptor of the pipe 707 combined with the front end log process 705 to a message function, and the message transmitted by the message function goes to the front end log process 705 instead of the back-end log process 716. If the front end log process 705 receives this message, the front end log process 705 will arrange this message to the log file 703 in a main system 511 (a).
[0094] In the desirable example, a message function detects the failure of a pipe 510 as follows. A process 503 uses a TCP-IP protocol and transmits a message to a pipe 701. In this protocol, only when a precedence message is received, the following message can be transmitted. Therefore, the message function used by the function in a library routine 507 transmits a message to a pipe 710 by transmitting two kinds of messages, i.e., the Shinsei message, and a dummy message. When a message function can transmit a dummy message, the Shinsei message reaches. If the failure of a system 513 occurs, the message transmitted through a pipe 710 does not reach and a dummy message cannot transmit it, either.
[0095] If backup system 513 is recovered, a monitor 719 will make the system call engine 715 and the back-end log process 716 restart, and will be notified to a monitor 717 after that. A monitor 717 is notified to the pipe process 711. The pipe process 711 acquires the file descriptor of a pipe 710, and terminates the front end log process 705. If the back-end log process 716 restarts by the system 513, a process 716 will acquire the copy of a log file 703 (a) from the kernel server 305 (a), and will add this copy to a log file 703 (b). Then, the system call engine 715 resumes activation of the message in a log file 703 (b).

[0096] The message function used by lib.3d is the same approach as the approach of acquiring the file descriptor about a pipe 707, and acquires the file descriptor about a pipe 710. Next, if it tries to use the file descriptor of a pipe 710 in order that a message function may transmit a message, this attempt will go wrong and, as for a message function, a pipe file descriptor will be again required from the pipe process 711. Shortly, a message function receives the file descriptor of a pipe 710, and is again connected to a back-end log file.
[0097] The remaining fault condition is processed as follows.
** The failure monitor 717 of the pipe process 722 detects this failure. The newly restarted process searches the connection with a pipe 710 from the process condition saved by WatchD. Other processes do not notice this failure and recovery.
[0098] ** The failure monitor 719 of the system call engine 715 detects this failure, and makes the system call engine 715 restart. The newly restarted system call engine 715 is recoverable from a foreign file to the precedence checkpoint condition with the checkpoint and recovery function which are given by libft. Other processes do not notice this failure and recovery.
[0099] ** The failure monitor 719 of the back-end log process 716 detects this failure, and makes the back-end log process 716 restart. A process 716 restores the condition from a checkpoint file. A monitor 719 notifies further that the back-end log process 716 was restarted to a monitor 717, and, subsequently to the pipe process 711, notifies a monitor 717. Then, a process 711 connects a pipe 710 to the new back-end log process 716. The next store of each application file and lib.3d acquires new connection from the pipe process 711.
[0100] ** As for the failure of the failure front end log process 705 of the front end log process 705, only the failure nascent state period of a system 513 exists. If a monitor 717 detects the failure of the front end log process 705, a monitor will be notified to the pipe process 711. Subsequently, the front end log process 705 is made to restart, and a pipe 708 is re-connected to the front end log process 705. The next store of an application program 509 goes wrong, consequently the message-sending function in lib.3d acquires the file descriptor about the new pipe 708 from the pipe process 711.
[0101] implementation [of the user level name space 405]: -- drawing 8 -11 -- in order to specify all a series of files from the file system supplied by the kernel server 305 (a) to the application process 503, the user level name space 405 can be used. Drawing 8 shows the relation between the name space 801 of the file system supplied by the kernel server 305 (a), and the user level name space 405 in the user level backup file system 701.
[0102] In name space 801, a file name is arranged in the shape of a tree. The file which forms the leaf (B, D, E, G, I, M, N) of the tree in drawing 8 contains data or a program. The remaining file is a list of other files. Such a file is called a directory. Even the name of the file which can specify all the files in name space 801 to the kernel server 305 (a) with the path name which starts by the root "/", and is specified with a path name from the root includes the name of all files. Therefore, the path name of a file "D" is /A/C/D and the path name of a file "L" is /J/K/L.
[0103] The user level backup file system 701 specifies the file which should back up by specifying the subtree of the name space 801 which includes a file. Subsequently, actuation to the file in the subtree which changes a file is performed to the backup file in backup system 513. In drawing 8 R> 8, it is chosen as what four subtrees, 803 (a), 803

(b), and 803 (c) should back up.
[0104] Therefore, modification to the data files D, E, G, I, and M or N in name space 801 will produce modification to the backup file of this data file. Similarly, modification to Directories C, F, H, and L will produce modification to these backup files. Since all the files in a subtree are backed up, the file which should back up is specified with the path name of the directory which is the root of a subtree in the user level name space 405. For example, a subtree 803 (a) is specified by a path name / A/C805 (a) in the user level name space 405.
[0105] Needless to say, the map of the user level name space 405 must be carried out also to the file system supplied with the system call engine 715. This is performed in the back-end map 517. As shown in drawing 9 , the back-end map 517 includes the entry about each open file in the user level name space 405. An entry consists of two parts, the user level name space information 903 and the backup system information 905, of 513.
[0106] The user level name space information 903 identifies the file in the user level name space 405. The backup system information 905 identifies the file in the system supplied by the kernel server 305 (b) corresponding to the file identified using user level name space information.
[0107] The back-end map 517 makes it possible to map the subtree of name space 801 in the subtree of the name space 907 of the file system which the kernel server 305 (b) supplied to the back-end log process 716 and the system call engine 715. This mapping is performed by mapping the path name of the root of the subtree of name space 801 in the path name of the root of a subtree where name space 907 corresponds.
[0108] The path name of the root is called the prefix of the path name of the file in a subtree. For example, the path name in a subtree 803 (a) has a prefix/A/C. The path name of the file E in a subtree 803 (a) is E. Subtree 909 / Z is made to correspond to a subtree 803 (a) in name space 907 by mapping a prefix/A/C in the prefix/Z of name space 907 from name space 801. After mapping is completed, modification of the file specified by the path name/A/C/E in name space 801 produces modification of the file specified by the path name/Z/E in name space 907.
[0109] detail [of the front end duplicate tree 505]: -- drawing 10 -- user level name space 405 is realized by the front end duplicate tree 505 in the desirable example. Drawing 1010 is a block diagram showing the front end duplicate tree 505 in a detail. Two main components of the front end duplicate tree 505 are RTREE1015 and the file descriptor cache 1027. The path name of the root of the subtree 803 which has the file by which RTREE1015 should be backed up is a linked list.
[0110] The file descriptor cache 1027 is an array which relates a file descriptor to a device and an entry (inode) identifier. Therefore, the form of this implementation is as a result of the fact that the file system offered by the UNIX operating system identifies a file by the entry (inode) of the identifier of a disk at which is attained to with an integer file descriptor and a file resides by three approaches, i.e., a path name, and the file in a UNIX file system table.
[0111] The file descriptor of a file is effective only in the process which opened the file again, only while this process opens this file. Although a UNIX file system table enables the translation between a device, and inode and the present file descriptor at a path name, a device, and the list between inode(s), the direct translation between a path name and the present file descriptor is not made possible.

[0112] If it explains to a detail further succeedingly, maxtry1003 and init1005 will be used for initialization of the front end duplicate tree 505. Before an initialization function gives up maxtry1003, it shows the count tried in order to set up a pipe 710 in backup system 513. init1005 shows whether the pipe was set up or not. The RPROP array 1009 is an array of the name 1011 of the actuation which can be performed by the front end duplicate tree 505.
[0113] RTREE PTR1013 is a linked list which includes the pointer of the element of the beginning of the RTREE list 1015, i.e., a certain element of each duplicate tree 803. Each element 1017 includes the pointer 1023 to the die length 1019 of the path name 1021 of the root of the duplicate tree 803, and a path name 1021, and the next element in a linked list. The connection server 1025 is a path name in the name space 801 of the pipe 710 to backup system 513.
[0114] The FD cache 1027 is the array of the file descriptor cache entry 1029. Many entries 1029 in this array exist like an available file descriptor to the application process 503. The index of the entry of the predetermined file descriptor in the FD cache 1027 is a file descriptor. While it is shown whether an entry 1029 has an effective entry actually and being opened by the file, the application process 503 includes the status flag which shows whether a child is borne or not.
[0115] An entry 1029 also includes the identifier 1101 of the file residence device in a main system 511, and the identifier of inode of the file in a main system 511. The effective entry 1029 of each file actually opened by the subtree 803 specified by the entry in RTREE1015 exists.
[0116] The detail back-end map 517 of the back-end map 517 consists of two parts, the path name map 1113 and the opening duplicate file list 1117. The path name map 1113 only carries out the map of the path name in the name space 801 of a main system 511 to the path name of the name space 907 of backup system 513. Each entry 1115 in a map establishes the relation between the front end path name 1117 and the back-end path name 1119.
[0117] The entry which maps the root of the subtree 803 in the front end name space 907 on the root of the subtree in name space 907 is included by the path name map 1113. The back-end path name 1119 is a part of backup system information 905. In the desirable example, these mapping is specified in a system configuration file.
[0118] The opening duplicate file list 1117 includes the entry 1119 of each file which the application process 503 opens actually in the duplicate tree 803. The user level name space information 903 in an entry 1119 includes the front end file identification child (FFID) 1105 and the front end path name (FP) 1106. The front end file identification child (FFID) 1105 consists of the device identifiers and inode identifiers of a file in a main system 511.
[0119] The front end path name (FP) 1106 is divided into the front end prefix (FPR) 1107 and the subtree path name 1108. The front end prefix (FPR) 1107 is a prefix of the subtree of the file in the front end name space 801. The subtree path name 1108 is a path name of the file in the subtree. The backup system information 905 in an entry 1117 consists of a back-end file descriptor 1111.
[0120] The back-end file descriptor 1111 is a file descriptor in the file system supplied by the kernel server 305 (b) about this file. In the desirable example, the back-end map 517 is realized as a hash table. This hash table can be accessed with both the front end file

identification child 1105 and the front end path name 1106.
[0121] The approach on which the approaches of creating actuation DS 505 and 517 including DS 505 and 517 and such DS are made to act by various file manipulation is explained. In the desirable example, the application process 503 is performed with the UNIX operating system which uses a Korn shell.
[0122] A process enables it, as for a Korn shell, to set up an ENV variable. An ENV variable specifies the file performed always, when a process calls a Korn shell. The file specified by the ENV variable in the application process 503 builds the front end duplicate table 505, and since it initializes, it contains information required for the application process 503.
[0123] If created, a table 505 becomes a part of address space of the application process 503, and is available to the child process of the application process 503. This child process is created by fork system call of a UNIX operating system. Therefore, a child process inherits the parents' environment. On the other hand, an exec system call gives a new environment to a child process.
[0124] In order to create the available front end duplicate tree 505 to the child of the application process 503 created by the exec system call, lib.3d has the exec function which copies the front end duplicate tree 505 to ENV available about a new process. For this reason, even if processes are other points and it is the case where those parents' address space is not inherited, lib.3d is available to this process. Even the pipe with a name or foreign file for passing the front end duplicate tree 505 to the child process created by exec can be used for other examples.
[0125] File manipulation is explained further. Actuation of the beginning of these actuation is mounting actuation. In a UNIX operating system, mounting adds the tree of a name to the name space of an operating system. In the desirable example, the version of mounting realized by lib.3d includes the mode made to add to the user level name space 405 by using the subtree of the front end name space 801 as the duplicate tree 805.
[0126] When mounting is used in this mode, a path name argument is a path name of the root of a subtree 803 added to the user level name space 405. This function adds a subtree 803 to the user level name space 405 by adding this entry to the duplicate tree 1015 by creating the duplicate tree entry 1017 about this path name. Moreover, unmounting actuation of deleting the duplicate tree entry 1017 which has the specified path name from the duplicate tree 1015 also exists.
[0127] When the application process 503 performs opening actuation to the file in the duplicate tree 805, the opening function in lib.3d creates the file descriptor cache entry 1029 about the newly opened file, and transmits an opening message to the back-end log process 716. This opening message contains the path name in a main system 511, device identifier, and inode identifier of the opened file.
[0128] If this message is performed with the system call engine 715, an entry 901 will be created on the back-end map 517. A path name 113 is used in order to discover the file in the back-end system 513 corresponding to the file opened with the main system 511. The corresponding file descriptor of a file is arranged at the back-end file descriptor 1111.
[0129] If opened by the file, in order to identify a file, a file identification child will be used for the file manipulation in a main system 511. The message for the actuation to the backup file in backup system 513 of corresponding uses a device identifier and an inode identifier, in order to identify a file. In order to perform such a message, the system call

engine 715 should just access the entry in the opening duplicate file list 1117 about the device and inode which were specified within the message. This entry includes the file descriptor 1111 required to perform backup system 513.
[0130] When the application process 503 closes the file in the duplicate tree 505, a lib.3d closing function is determined from Status field 1033 about whether the child process is using the file. When the child process is not using the file, a closing function makes an invalid the file descriptor cache entry 1029 about the file in the duplicate tree 505, and transmits the closing message containing a device identifier and an inode identifier to backup system 513.
[0131] When the system call engine 715 performs this message, in order to find the entry 1119 of this file, a device and an inode identifier are used. Subsequently, the file in backup system 513 is closed using the back-end file descriptor 1111 for identifying a file, and, finally an entry 1119 is deleted from the opening duplicate file list 1117.
[0132] As mentioned above, although an example of the formation approach of a user level backup file system and operation has been explained to a detail, it is clear to this contractor that various examples of modification of the user level backup file system of this invention are realizable. For example, it reflects that a desirable example is realized by the system which performs a UNIX operating system, and much details of a desirable example are realized by the system which performs a UNIX operating system. other operating systems -- it is, and these details are changed when realizing.
[0133] Similarly, the common library system used with the operating system of Sun is used for a desirable example. Other examples use the configuration of others of the library linked dynamically. Furthermore, the thing and functional target which were indicated by this specification can also be made to realize the DS indicated on these specifications, and a message by many of approaches of equivalent others.
[0134] Furthermore, it passes over a user level backup file system only to an example of the operation of the alternate library where it was dynamically linked for changing the behavior of a process, without changing the application program currently performed according to the process. This same technique can be used also in order to define different user level name space for processes from the name space given to the process by the operating system, and not only file backup but other services can be used for it also in order for user level name space or an operating system to add to the name of which entity of the name space given to a process.

[Translation done.]

DESCRIPTION OF DRAWINGS

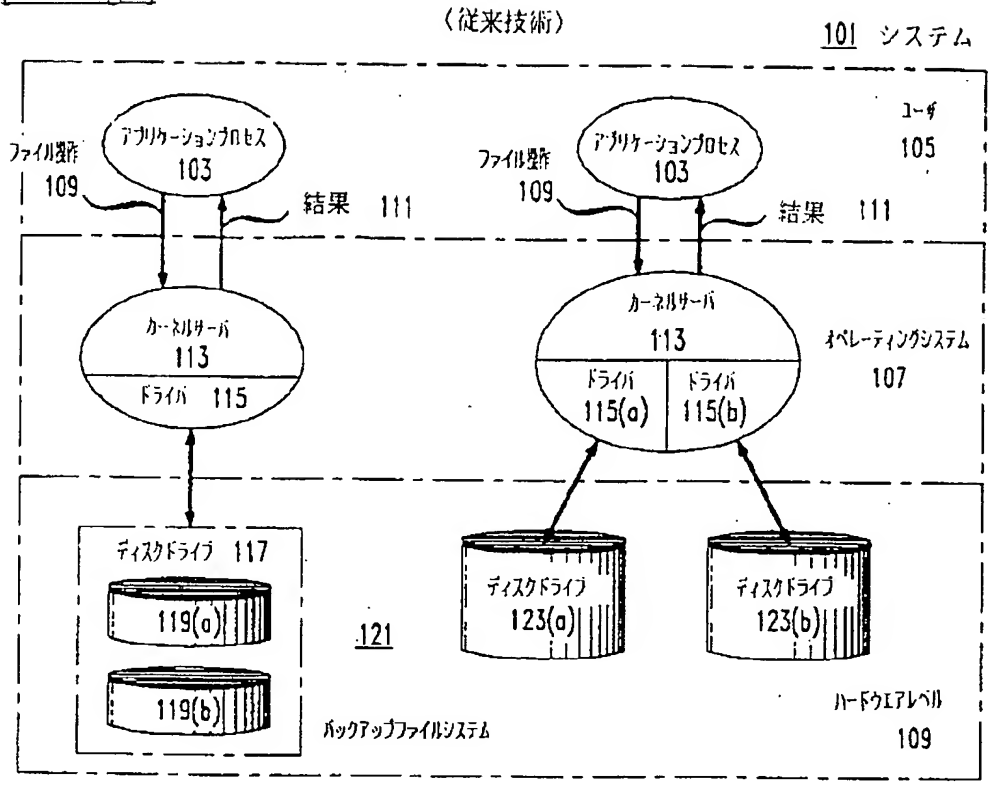
[Brief Description of the Drawings]
[Drawing 1] It is the block diagram of the conventional backup file system.
[Drawing 2] It is the mimetic diagram showing how a library redefines the interface of a user program a .
[Drawing 3] It is the block diagram showing how to redefine an operating system interface using the library linked dynamically.

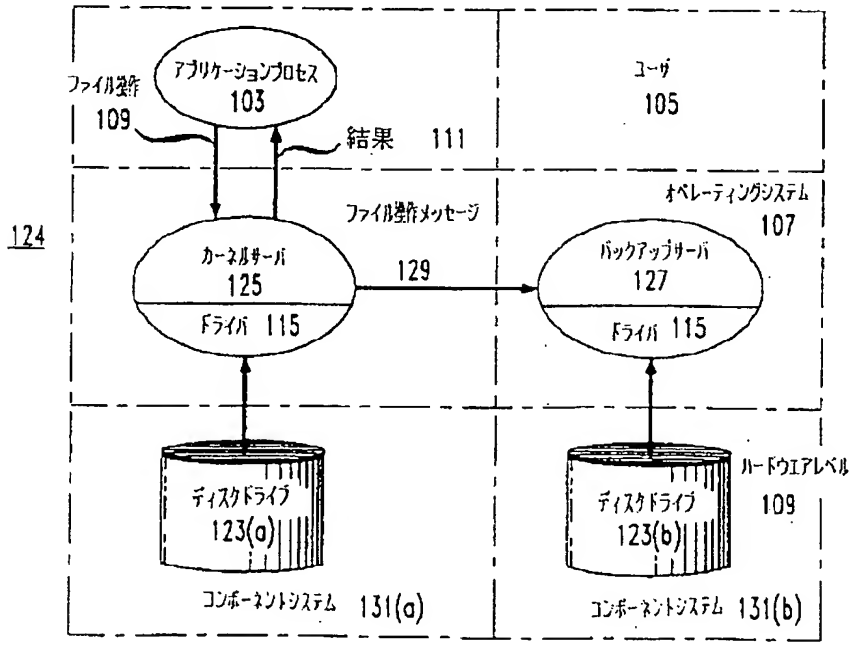
[Drawing 4] It is the block diagram showing how to supply user level name space using the library linked dynamically.
[Drawing 5] It is the block diagram of the user level backup file system which uses the library linked dynamically.
[Drawing 6] It is the schematic diagram of the routine in the library linked dynamically.
[Drawing 7] It is the block diagram of the desirable example of a user level backup file system.
[Drawing 8] It is the block diagram showing the relation of the name space and user level name space which are supplied by the kernel server (a).
[Drawing 9] It is the block diagram showing the relation between user level name space and the name space supplied by the kernel server (b).
[Drawing 10] It is the block diagram showing the structure of the front end duplicate tree 505.
[Drawing 11] It is the block diagram showing the structure of the back-end map 517.
[Description of Notations]
101 The Conventional Backup File System
103 Application Process
105 User
107 Operating System
113 Kernel Server
115 Driver
117 Disk Drive
305 Kernel Server
307 User Process
309 Application Program
315,321 Operating system library
403 Operating System Library
405 User Level Name Space
409 User Process
501 User Level Backup File System
503 Application Process
505 Front End Duplicate Tree
507 lib.3d
509 Application Program
511 Main System
513 Backup System
515 Back-end Map
517 Back-end Map
701 User Level Backup File System
703 Log File
705 Front End Log Process
707,709,710 Pipe
711 Pipe Process
715 System Call Engine
716 Back-end Log Process
717,719 Monitor

801 File System Name Space
903 User Level Name Space Information (ULNI)
905 Backup System Information (BSI)
907 Name Space
909 Subtree

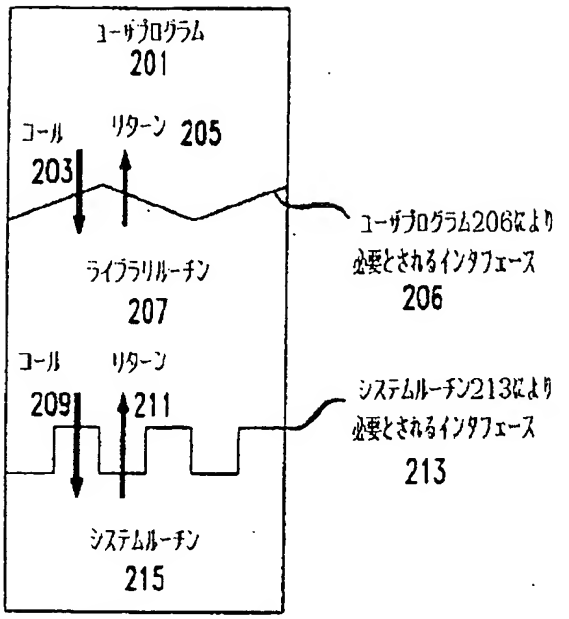
DRAWINGS

[Drawing 1]

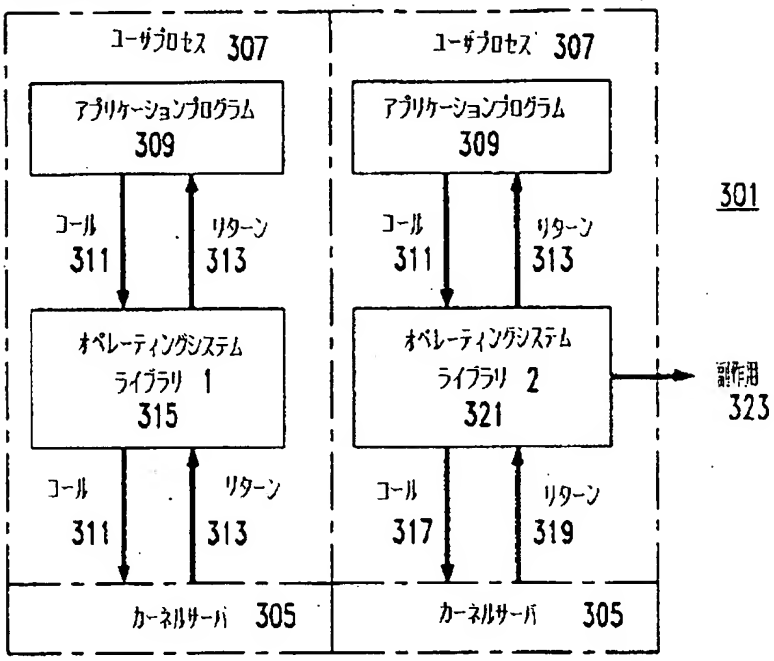




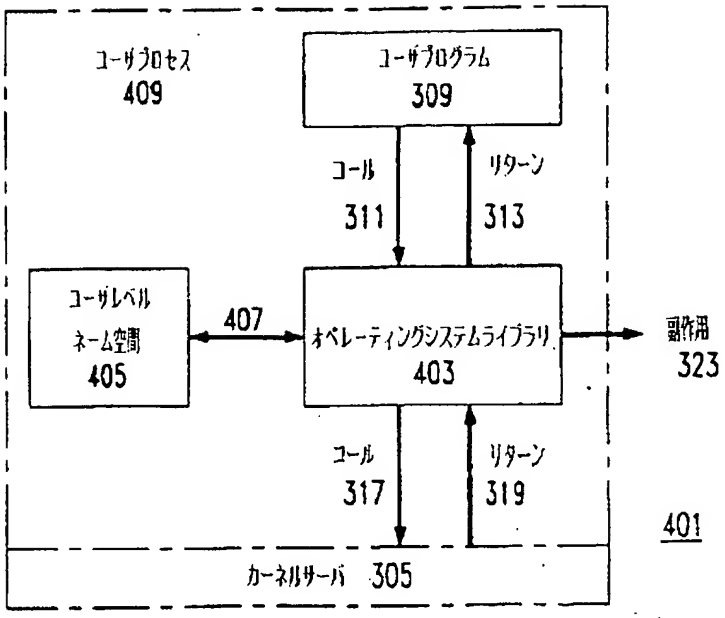
[Drawing 2]



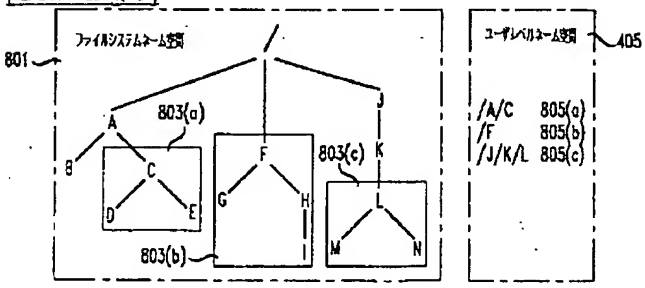
[Drawing 3]



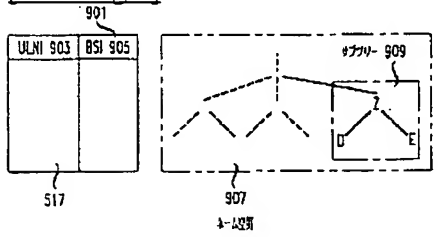
[Drawing 4]



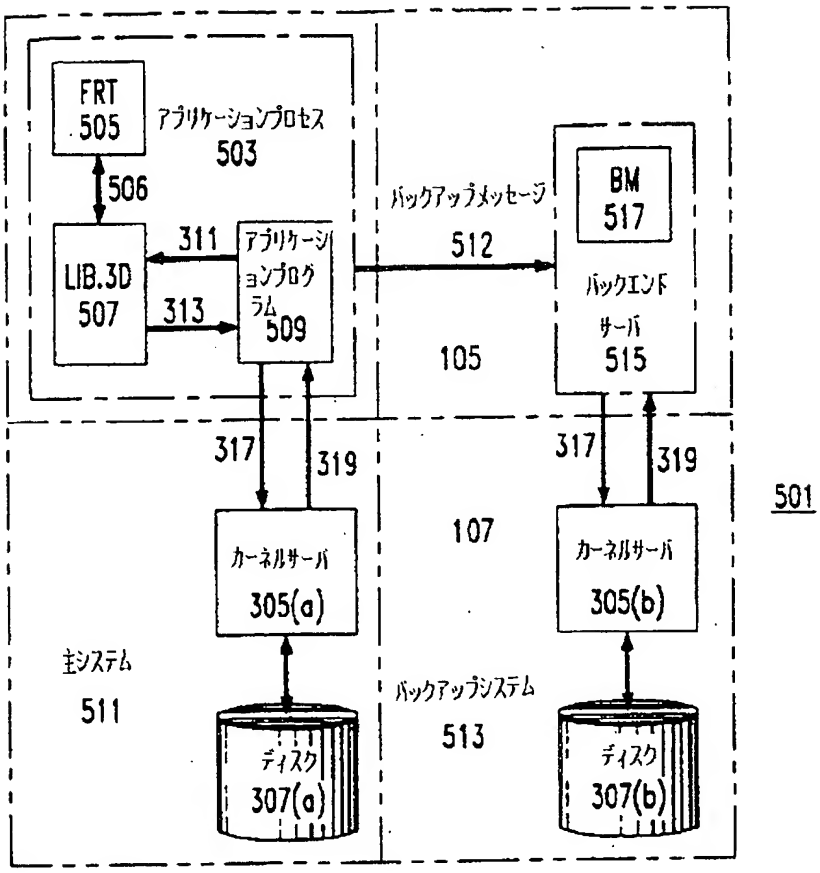
[Drawing 8]



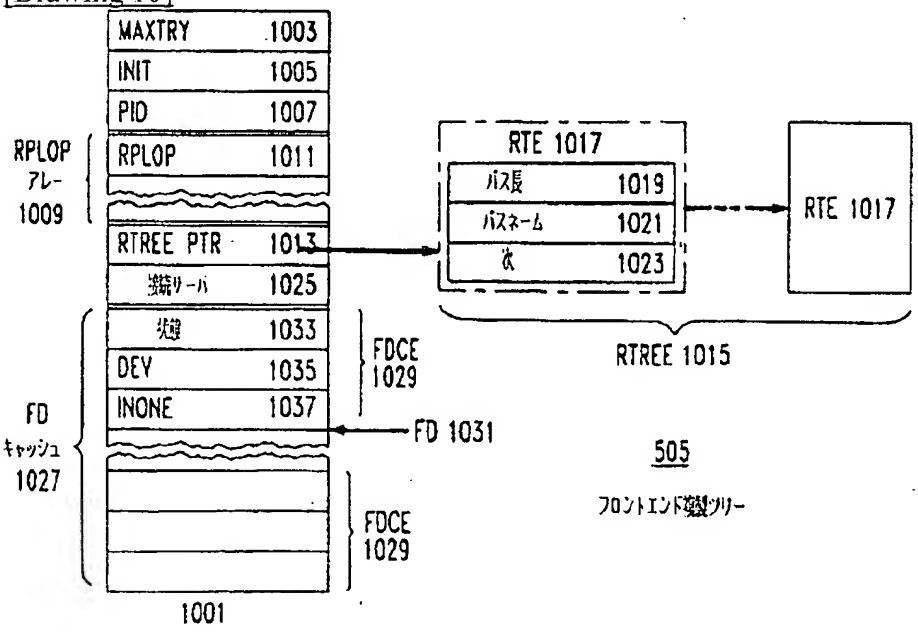
[Drawing 9]



[Drawing 5]



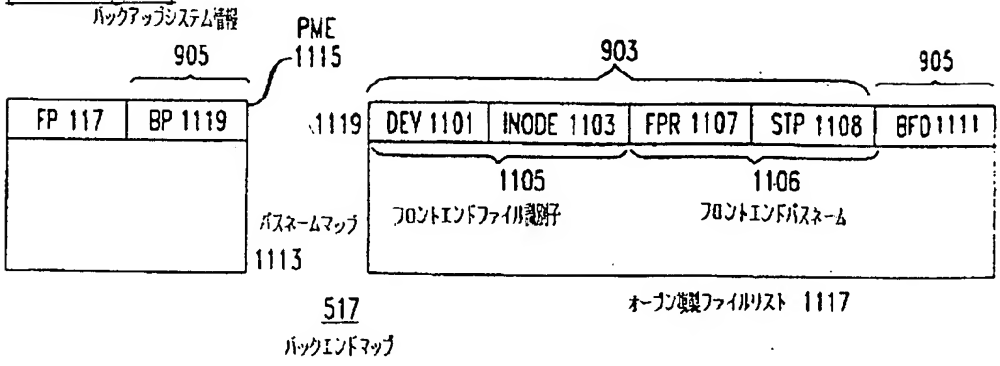
[Drawing 10]



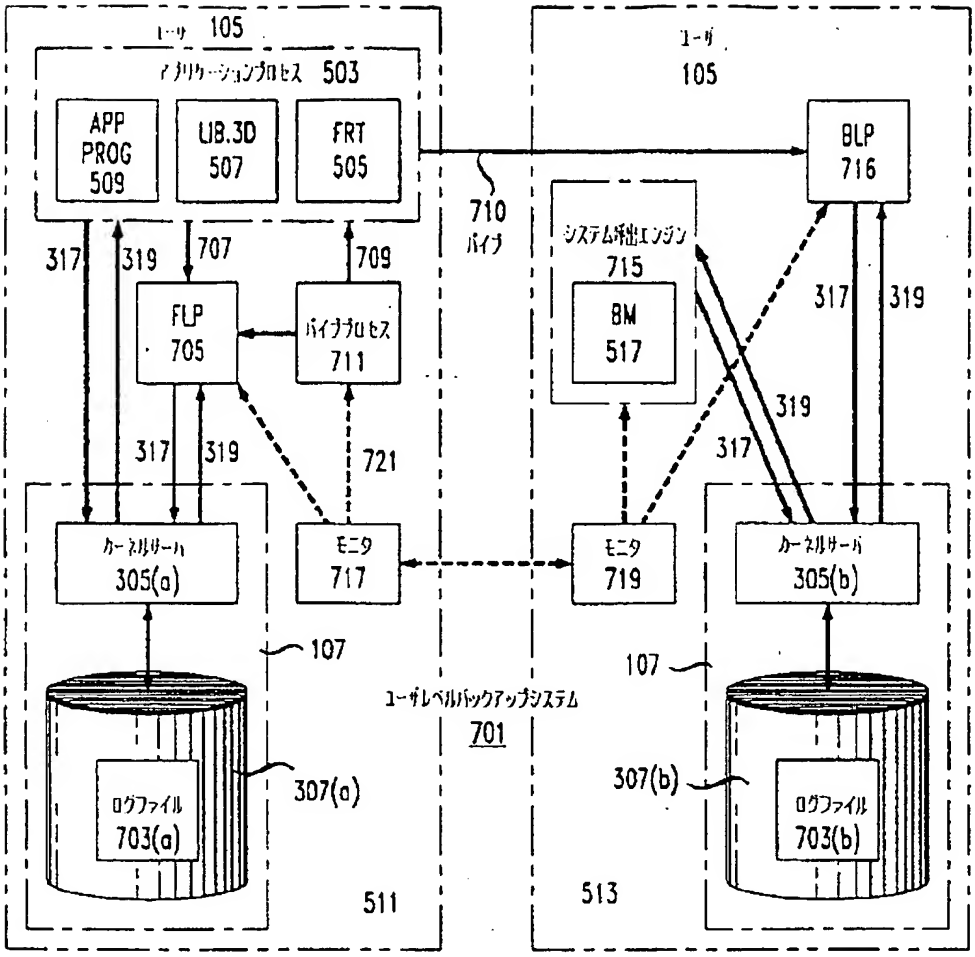
[Drawing 6]

```
                603                                   605
初期化<アプリケーションプログラム-ファイル操作>(<アプリケーションコード-ファイル操作-引数>)
{
    607                 609
  IF<サーバーファイル操作>(<サーバーファイル操作-引数>)
          613           614
      IF IN-複製ツリー(<ファイル-引数>)
          615           617
        { 送信-メッセージ(<メッセージ-引数)
          リターン(成功)
        }
      ELSE
          リターン(成功)
  ELSE
      リターン(成功)
}
```

[Drawing 11]



[Drawing 7]


105
アプリケーションプロセス 503
APP PROG 509
UB.3D 507
FRT 505
710 パイプ
105
BLP 716
システム呼出エンジン 715
BM 517
317
319
707
709
FLP 705
パイププロセス 711
721
317
319
317
319
319
317
カーネルサーバ 305(a)
モニタ 717
モニタ 719
カーネルサーバ 305(b)
107
107
ユーザレベルバックアップシステム 701
307(a)
307(b)
ログファイル 703(a)
ログファイル 703(b)
511
513

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平7－13751

(43)公開日　平成７年(1995)１月17日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ	技術表示箇所
Ｇ０６Ｆ　9/06	４１０　Ｈ	9367－5B		
12/00	５３１　Ｄ	8944－5B		

審査請求　未請求　請求項の数２　ＦＤ　（全 19 頁）

(21)出願番号　特願平6－155409

(22)出願日　平成６年(1994)６月15日

(31)優先権主張番号　０８００３７

(32)優先日　1993年６月18日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390035493
エイ・ティ・アンド・ティ・コーポレーション
ＡＴ&Ｔ　ＣＯＲＰ.
アメリカ合衆国　10013－2412　ニューヨーク　ニューヨーク　アヴェニュー　オブ　ジ　アメリカズ　32

(72)発明者　グレン　ステファン　フアウラー
アメリカ合衆国、07076　ニュージャージー、スコッチ　プレインズ、ライド　プレイス　2322

(74)代理人　弁理士　三俣　弘文

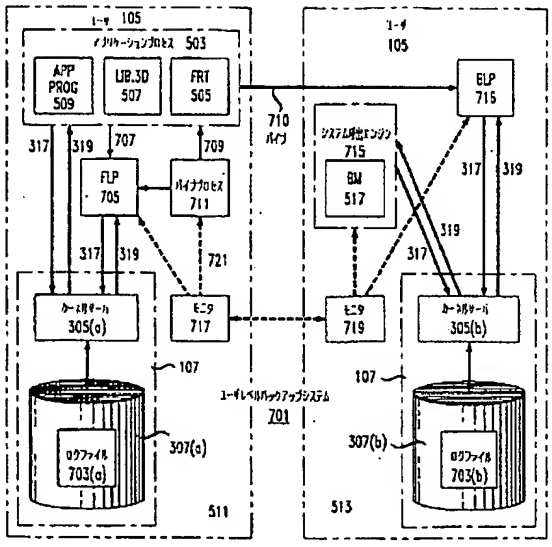
最終頁に続く

(54)【発明の名称】　ファイル・システムの副作用の選択的追加装置

(57)【要約】　（修正有）

【目的】　ＡＰ、ＯＳ又はハードウエアを変更することなしに使用できるバックアップファイルシステム（ＢＦＳ）を提供する。

【構成】　ＢＦＳは、主コンピュータシステム及びＢＣＳに対する動的にリンク可能な代替ライブラリ（ＤＬＲＬ）及びユーザレベルプロセス（ＵＬＰ）により実現される。ＤＬＲＬはファイル操作（ＦＯ）の標準的なＤＬＬと同じＩＦを有する。ＤＬＲＬにおけるファンクションはＤＬＬにおけるファンクションと同じＦＯを行う。このファンクションは現に行われている操作を指定するメッセージ（ＭＳ）をＢＣＳのＵＬＰに送る。ＵＬＰは、ＭＳで指定された操作をＢＣＳにおけるバックアップファイル（ＢＦ）に対して行う。被ＢＦはＰＦＳのネーム空間（ＮＳ）のサブツリー（ＳＴ）を識別することにより指定される。ＳＴはユーザ定義ＮＳを形成する。ＢＣＳ及びＢＦＳのプロセスをモニタし、必要に応じて障害を処理するモニタプロセスを使用することにより、ＢＦＳは誤り許容にされる。



【特許請求の範囲】

【請求項１】 動的にリンクされるライブラリにおけるファンクションを実行することにより行われるエンティティに対する操作に副作用を選択的に追加する装置において、

追加すべき副作用の表示をエンティティに対応付ける手段と，動的にリンクされるライブラリを代替し、かつ、代替ファンクションを有する動的にリンクされる代替ライブラリとからなり、

前記代替ファンクションは、動的にリンクされるライブラリにおけるファンクションと同じ方法で操作を行い、そして、更に、副作用がエンティティについて行われるべきものか否か表示対応付け手段から決定し、行われるべきものと表示対応付け手段が示した場合、該副作用を行うことを特徴とするファイル・システムの副作用の選択的追加装置。

【請求項２】 第１の動的にリンクされるライブラリからのファンクションを用いて第１のファイルシステムにおける第１のファイルについて動作するアプリケーションプログラムを実行する第１のユーザプロセスと，第２のファイルシステムにおいて動作する第２のユーザプロセスと，第１の動的にリンクされるライブラリを代替し、かつ、第１の動的にリンクされるライブラリからのファンクションの代替ファンクションを有する第２の動的にリンクされるライブラリ、該代替ファンクションは第１のライブラリからのファンクションと同じ方法で第１のファイルシステムにおけるファイルについて動作し、更に、第２のユーザプロセスに対する操作を指定するメッセージを送信する，及び第２のユーザプロセスにより実行可能な、第２のファイルシステムにおける第２のファイルに関するメッセージにより指定される操作を行うことにより該メッセージに応答する手段と，からなるファイル複製システム。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明はコンピュータシステムに関する。更に詳細には、本発明はオペレーティングシステムに対して行われた呼出の効果を修正するために、システムのアプリケーションプログラムレベルで使用する手法に関する。

【０００２】

【従来の技術】コンピュータシステムは階層化されている。典型的なシステムでは、階層は、最下位層のハードウエア（例えば、プロセッサ，メモリ，大容量記憶装置及びこれらの部品をリンクさせる通信媒体）を包含する。次の層はオペレーティングシステムである。オペレーティングシステムはハードウエアの操作を制御し、一連の論理デバイスを定義する。論理デバイスにおける操作はオペレーティングシステムにより制御されるハードウエアにおける操作になる。オペレーティングシステムにより供給される論理デバイスはオペレーティングシステムよりも上位層におけるプログラムにより使用される。これらの階層はシステムのユーザレベルと呼ばれている。

【０００３】コンピュータシステムの設計における重要な問題点はシステムにより実行される操作が定義されるレベルである。例えば、オペレーティングシステムはアプリケーションプログラムレベルに豊富な一連の論理デバイスを供給するか、又は小規模な一連の基本的論理デバイスを供給し、操作を行うために必要に応じてユーザのアプリケーションプログラムが基本的デバイスを結合することを期待する。これはハードウエアの場合も全く同様である。各レベルにおける設計上の二律背反は同じである。

【０００４】豊富な一連の論理デバイスは次のレベルにおけるプログラミングを簡単にするが、論理デバイスを供給するレベルの複雑性が増大し、その結果、システムの全体的な柔軟性は低下する。特に、システムの下位層で複雑な操作が行われる場合、これらの操作は、上位層で利用可能な情報の恩恵無しに行わなければならない。

【０００５】ここに述べた設計上の問題点は、エーシーエム・トランザクションズ・オン・コンピュータ・システムズ(ACM Transactions on Computer Systems)，第２巻，第４号（１９８４年１１月発行），２７７～２８８頁に掲載された、ジェー・エッチ・サルザー(J.H.Saltzer)らの「システム設計におけるエンド・ツー・エンド問題」と題する論文中に詳細に説明されている。

【０００６】二律背反の一例はバックアップファイルシステムの設計によりもたらされる。バックアップファイルシステムは別のファイルシステムの一部又は全部のバックアップを与える。これはバックアップされているシステムなので、バックアップファイルシステムはファイル内容を保存するでけでなく、バックアップされているシステムの一部分の中の既知のファイル名も保存する。ファイルについて行われる操作はバックアップファイルシステムにおけるファイルについても繰り返すことが望ましい場合が多々ある。

【０００７】バックアップファイルシステムが存在する場合、オリジナルファイルシステムが破壊されるか又は動作不能になっても、失われる情報は皆無である。更に、バックアップファイルシステムは内容と共に名称（ネーム）も保存するので、バックアップファイルシステムは故障したファイルシステムで使用されていたプログラムにより即座に使用することができる。言うまでもなく、アプリケーションプログラムは常にそれ自体のバックアップファイルシステムを作製することができるが、バックアップファイルシステム問題の好ましい解決法は常に、アプリケーションの変更なしにバックアップを行うことができる方法である。

【０００８】図１はバックアップファイルシステムの設

計に対する従来のアプローチを示す。多重処理コンピュータシステムでは、アプリケーションプロセスで実行するプログラムにより特定される操作に応答して、ファイルシステムはファイルに対する操作を行う。ファイルシステム自体は少なくとも一つのオペレーティングシステムプロセス及びハードウエア（例えば、ディスクドライブ）を用いて実現される。

【０００９】アプリケーションプロセスとオペレーティングシステムプロセスとの間の関係はクライアントとサーバの関係である。アプリケーションプロセスは、サーバプロセスが操作を行い、その操作の結果をアプリケーションプロセスに返送することを要求する。従って、システム１０１では、アプリケーションプロセス１０３は、カーネルサーバ１１３がファイル操作１０９を行うことを要求する。このため、カーネルサーバ１０９は操作を行い、そして結果１１１を返送する。

【００１０】言うまでもなく、カーネルサーバ１０９はディスクドライブ１１７でデータを変更することにより操作を行う。ディスクドライブ１１７とカーネルサーバ１１３との間のインターフェースは、所定のタイプのディスクドライブ１１７に対して特有のドライバ１１５と呼ばれるカーネルサーバ１１３の部品である。

【００１１】従来の技術では、バックアップファイルシステムはハードウエアレベル１１９とオペレーティングシステムレベル１０７において実現されていた。図１のシステム１０１ではハードウエアレベル１０９において実現されている。ディスクドライブ１１７はサーバ１１３に対して通常のディスクドライブとして出現するが、ミラーディスク１１９（ａ）及び１１９（ｂ）を包含する。

【００１２】各ファイルは両方のディスクのコピーを有し、ファイルを変更する操作は各ディスクのコピーに対して行われる。従って、各ディスクはファイルシステムのコピーを包含する。ディスクの一方が壊れても、他のディスクのシステムは依然として利用可能である。言うまでもなく、システム１０１の特別な欠点は特定のハードウエアを必要とすることである。

【００１３】バックアップファイルシステム１２１及び１２３はオペレーティングシステムレベルで実現される。システム１２１は２つの非ミラーディスク１２３（ａ）及び１２３（ｂ）を有する。各ディスクは別個のドライバ１１５（ａ）及び１１５（ｂ）を有する。サーバ１１３がファイルの作成，削除又はその他の変更などのファイル操作を行う場合、ドライバ１１５（ａ）はドライブ１２３（ａ）にこの操作を行わせ、ドライバ１１５（ｂ）はドライブ１２３（ｂ）にこの操作を行わせる。

【００１４】結果的に、ドライブ１２３（ａ）とドライブ１２３（ｂ）にファイルシステムの同一コピーが存在する。システム１２１はもはや特別なハードウエアを必要としないが、変更されたオペレーティングシステムは依然として必要である。更に、変更はドライバ１１５（ａ）及び（ｂ）について行わなければならないので、この変更はカーネルサーバ１１３の最下位層で行わなければならない。

【００１５】バックアップファイルシステム１２３も２つの非ミラーディスク１２３（ａ）及び１２３（ｂ）を有するが、各ディスクは別個のサーバ１２５及び１２７により駆動される。サーバ１２５は第１のディスク１２３（ａ）についてファイル操作１０９を行う。このファイル操作がファイルの作成，削除又はその他の変更である場合、サーバ１２５はファイル操作メッセージもバックアップサーバ１２７に送信する。そして、その後、同じファイル操作を行う。

【００１６】その結果、アプリケーションプロセス１０３により現に使用されているファイルシステムのコピーが両方のディスクドライブ１２３（ａ）及び１２３（ｂ）に存在する。バックアップファイルシステム１２３は一般的に、多数のコンポーネントコンピュータシステム１３１を含む分散コンピュータシステムで実現される。このようなシステムでは，アプリケーションプロセス１０３，カーネルサーバ１２５及びディスクドライブ１２３は一般的に、１個のコンポーネントシステム１３１（ａ）に存在するが、バックアップサーバ１２７及びディスクドライブ１２３（ｂ）は別のコンポーネントシステム１３１（ｂ）に存在する。

【００１７】システム１２１と同様に、システム１２３はオペレーティングシステムの変更を必要とするが、アプリケーションプロセス１０３に関するファイルを変更するときに常にメッセージをサーバ１２７に送るようにサーバ１２５を変更するだけでよい。バックアップサーバ１２７は、コンポーネントシステム１３１（ｂ）について他のプロセスからのファイル操作１０９に応答するのと全く同じ方法でこのメッセージに応答する。

【００１８】システム１０１，１２１及び１２３は何れもバックアップファイルの作成に有効であるが、これらは全て重大な欠陥を有する。第１に、これらのシステムは特別なハードウエア又はオペレーティングシステムの変更のどちらかを必要とし、その結果、携帯型にはならない。

【００１９】これらのシステムが実行されるべきコンピュータシステムは特別なハードウエア又は特別なオペレーティングシステムを有しない場合、これらのコンポーネントを獲得しなければならない。更に、システム１０１，１２１又は１２３が一旦使用状態になると、全ての交替用コンピュータシステムは特別なハードウエア又は特別なオペレーティングシステムを有しなければならない。

【００２０】第２に、システム１０１，１２１及び１２３はハードウエアレベル１０９又はオペレーティングシ

ステムレベル１０７で動作するので、全てのファイルがこれらのシステムに均等に出現し、これらのシステムはアプリケーションプロセス１０３により変更された全てのファイルをバックアップするだけである。

【００２１】しかし、一般的に、全てのファイルをバックアップする必要は無い。ＵＮＩＸ（登録商標）オペレーティングシステムにおけるファイルを分析したところ、ファイルシステム中のファイルのうちの５０～６０％は３分間未満の寿命しか有しない一時的作業ファイルであることが判明した。多くの場合、これらのファイルのうちの殆どのものがバックアップする必要の無いものである。

【００２２】しかし、カーネルサーバ１１３に対してこれらのファイルを識別する方法が存在しない。言うまでもなく、不必要なバックアップを行うことはシステム計算リソース及びシステム記憶リソースの浪費である。

【００２３】不必要なバックアップは前記の一般的問題の特別な事例である。ユーザから操作が行われれば行われるほど、この操作に関する情報はシステムにとって利用されなくなる。バックアップの事例で示されるように、結果は殆ど例外無く、システムリソースの非効率な使用ということになる。

【００２４】

【発明が解決しようとする課題】従って、本発明の目的は、ユーザ情報が有用な操作をアプリケーションプログラムの変更無しにユーザレベルで実行できるシステムを提供することである。

【００２５】本発明の別の目的は、ユーザレベルで動作し、従って、携帯することができ、かつ、バックアップされるべきファイルに関するユーザ情報の利点を利用できるバックアップファイルシステムを提供することである。

【００２６】

【課題を解決するための手段】前記の問題点は本発明のユーザレベルバックアップファイルシステムにより解決される。本発明のバックアップファイルシステムは、カーネルサーバにファイルコマンドを供給する標準的な一連のライブラリルーチンを、標準的な一連のライブラリルーチンの操作をエミュレートし、また、バックアップファイルシステムも処理する一連のライブラリルーチンに置き換える。アプリケーションプログラムが、ファイルを変更する操作を指定する場合、この操作のための代替ライブラリルーチンはこの操作を実行する。

【００２７】代替ライブラリルーチンは２つの行為を行う。すなわち、カーネルサーバに指定操作を実行させ、そして、カーネルサーバにより実際に行われる操作を指定するメッセージを別のコンピュータシステムにおけるユーザレベルバックエンドサーバに供給する。次いで、ユーザレベルバックエンドサーバは、このメッセージで指定されたファイル操作を他のコンピュータシステムにおけるカーネルサーバに供給する。

【００２８】従って、バックアップファイルの作成及び保守は代替ライブラリルーチンにより行われるファイル操作の副作用になる。バックアップファイルシステムの全てのコンポーネントはユーザレベルに存在し、そして、代替ライブラリルーチンはアプリケーションプログラムにより元々使用されるライブラリをエミュレートするので、このシステムは、ハードウエア，オペレーティングシステム又はアプリケーションプログラムを変更すること無しに使用することができ、更に、代替ライブラリでルーチンを実行できる全てのシステムで使用することができる。

【００２９】本発明のユーザレベルバックアップファイルシステムの別の利点は、どのファイルをバックアップすべきか指定するためにクライアントプロセス３０３を行うことである。この指定は複製ツリーデータ構造により行われる。この複製ツリーデータ構造は、バックアップすべきファイルを包含するクライアント３０３にアクセス可能なファイルシステムの部分を指定する。

【００３０】ファイル操作をエミュレートするライブラリルーチン内のコードは、ライブラリルーチンが操作を実行する際のファイルがバックアップされるべきものか否か決定するために、複製ツリーデータ構造をチェックする。バックアップされるべきものであり、かつ、操作がファイルを変更する場合、ファイル操作を包含するメッセージはバックエンドサーバプロセスに送信される。

【００３１】コンピュータシステムの下位層により提供される相関性のユーザレベル再定義を可能にするために、一般的に、或る種のエンティティに対する操作の副作用を指定するライブラリルーチンと、どのエンティティが副作用により悪影響を受けているか指定するデータ構造の組合わせが使用される。例えば、ユーザレベル論理ファイルシステムを定義し、オペレーティングシステムにより提供されるファイルシステムにユーザレベル論理ファイルシステムをマップするのに、この手法を使用することができる。

【００３２】

【実施例】以下、図面を参照しながら本発明を具体的に説明する。

【００３３】ライブラリの使用によるインターフェース変更：図２

前記で説明したように、コンピュータシステムは階層化されている。各層は次の上位層に対してインターフェースを形成する。上位層は、下位層のインターフェースにより必要とされるに応じて、下位層により実行されるべき操作を指定する。上位層が下位層により必要とされるインターフェースに適合しない場合、上位層と下位層との間にアダプター層を追加しなければならない。

【００３４】アダプター層の目的は、上層レベルにより期待されるインターフェースに従って作成された操作仕

様を下位層のインターフェースにより必要とされる操作仕様に翻訳することである。この手法を使用すれば、例えば、ＭＳＤＯＳオペレーティングシステムを実行するＰＣを、そのユーザに、ＵＮＩＸオペレーティングシステムを実行するコンピュータであるかのように思わせることができる。

【００３５】非常に多数のアプリケーションプログラムによりアダプター層が必要とされる場合、これはしばしば、一連のライブラリルーチンとして実現される。この名称が示唆するように、ライブラリルーチンは、コンピュータシステムのサブシステムのメーカーがアプリケーションプログラムと共に使用するためにコンピュータシステムのユーザに提供するルーチンである。

【００３６】図２はアダプター層を作成するためのライブラリルーチンの使用方法を示す。ユーザプログラム２０１は次の階層（この場合、一連のシステムルーチンである）に対してインターフェース２０６を有する。しかし、ユーザプログラム２０１が使用されるべきコンピュータシステムにおけるシステムルーチンはインターフェース２１３を有する。インターフェース２０６とインターフェース２１３との間の相違は、図２において、各インターフェースを示す実線の形状が異なることにより示される。

【００３７】アダプター層はライブラリルーチン２０７からなる。ライブラリルーチン２０７は、ユーザプログラム２０１により必要とされる上位の次層用のインターフェース２０６と、システムルーチン２０５により必要とされる下位の次層用のインターフェース２１３を有する。実際、インターフェースはファンクション（関数又は機能）呼出からなる。ライブラリルーチン２０７におけるルーチンは、インターフェース２１３により必要とされるファンクション呼出を生成することにより、インターフェース２０６により必要とされるファンクション呼出２０３に応答して動作し、ファンクション呼出２０３により指定される操作を行う。

【００３８】システムルーチン２１５が終了すると、矢線２１１で示されるように、システムルーチンはその実行結果をライブラリルーチン２０７に返送し、次いで、ライブラリルーチン２０７はリターン２０５で示されるように、結果をユーザプログラム２０１に返送する。

【００３９】動的にリンクされたライブラリルーチンの使用によるインターフェース再定義

ユーザプログラム２０１の実行コードを生成する場合、ライブラリルーチンをユーザプログラム２０１にリンクさせなければならないという事実により、従来のシステムでは、インターフェースの再定義に関するライブラリルーチンの有用性は低かった。

【００４０】ここでいう“リンキング”とは、ユーザプログラム２０１におけるライブラリルーチンの呼出をライブラリルーチン２０７のコピー内のライブラリルーチンの位置に関連させる処理のことである。実行可能コードが生成された場合、リンキングを行わなければならなかったので、実行可能コードのコピーしか有しないユーザは、或る一連のライブラリルーチン２０７を別の一連のライブラリルーチン２０７に置き換えることは不可能であった。

【００４１】ライブラリルーチンをユーザプログラムに動的にリンクさせることのできるコンピュータシステムが本発明により開発された。このようなコンピュータシステムでは、ユーザプログラムを実行するプロセスが、実行前にコンピュータシステムのメモリにロードされる場合に、リンキングが行われる。

【００４２】動的リンキングにより、ユーザプログラムのオブジェクトコードを変更すること無しに、或る一連のライブラリルーチンを別の一連のライブラリルーチンに置き換えることができ、これにより、ユーザプログラムが操作されるシステムの挙動を変更することができる。動的リンキングに関する説明は、米国、カリフォルニア州のマウンテンビューに所在するサンマイクロシステムズ社から１９８８年５月に発行された「共用ライブラリ(Shared Libraries)」に記載されている。

【００４３】図３は、動的リンキングを用いてシステムの挙動を変更する方法を示す。システム１３０１では、ユーザプロセス３０７は、オペレーティングシステムライブラリ１３１５が動的に結合されたアプリケーションプログラム３０９を実行する。オペレーティングシステムライブラリ１３１５は、コール３１１で指定される操作を行うために、コール３１１及びリターン３１３並びにカーネルサーバ３０５へのユーザコール３１７及びカーネルサーバ３０５からのリターン３１９により示されるアプリケーションプログラム３０９へインターフェースを供給する。

【００４４】システム２では、ユーザプロセス３０７は同じアプリケーションプログラム３０９を実行し、同じカーネルサーバ３０５を使用する。しかし、この場合、オペレーティングシステムライブラリ１３１５の代わりにオペレーティングシステムライブラリ２３２１が使用される。オペレーティングシステムライブラリ２３２１はオペレーティングシステムライブラリ１３１５が行うことは全て行う。しかし、オペレーティングシステムライブラリ２３２１は更に、副作用３２３を生成する。

【００４５】従って、システム３１０を、システム３０１と同様に挙動するが副作用３２３も生成するシステム３０３に変換するために必要なことは、オペレーティングシステムライブラリ１３１５の代わりに、オペレーティングシステムライブラリ２３２１をユーザプログラム３０９に動的にリンクさせることだけである。

【００４６】動的にリンクされたライブラリを使用することによるユーザレベルネーム空間の作成

図４は、動的にリンクされたオペレーティングシステム

ライブラリ４０３を用いてユーザレベルネーム空間を作成する方法及びこのユーザレベルネーム空間を用いて副作用２０３を制御する方法を示す。ファンクション，ファイル及びデバイスなどのようなコンピュータシステムにおけるエンティティはネームによりプログラム中で引用される。

【００４７】従って、コンピュータシステムのネーム空間のファンクションは、プログラム中で使用されるネームを、このネームにより示されるエンティティに関連させることである。従来のコンピュータシステムでは、ユーザプログラムにより使用されるネーム空間はオペレーティングシステムにより作成され、保持される。本発明のシステム４０１では、オペレーティングシステムライブラリ４０３がユーザプロセス４０９用の１個以上のユーザレベルネーム空間を作成し、保持する。

【００４８】ユーザレベルネーム空間４０５をライブラリルーチン４０３により使用できる一つの方法は、カーネルサーバ３０５によりユーザプログラム３０９が供給されるファイルシステムと挙動，構造又はこの両方ともが異なるユーザレベル論理ファイルシステムを作成することである。この論理ファイルシステムはその後、副作用を制御するために使用できる。

【００４９】例えば、システム４０１がバックアップファイルシステムディスクある場合、副作用３２３はバックアップファイルシステムを生成するのに必要なものであり、ユーザレベルネーム空間４０５は、カーネルサーバ３０５により供給されたファイルシステム中のどのファイルをバックアップファイルシステムにおいてバックアップすべきか指定する。図４から明らかなように、ユーザレベルネーム空間はユーザプロセス４０９の環境の一部である。

【００５０】ユーザレベルバックアップファイルシステムの概観：図５及び図６

アプリケーションプログラムを実行するアプリケーションプロセスにより変更されたファイルのうちの特定のファイルだけを自動的にバックアップするユーザレベルバックアップファイルシステムを形成するために、前記の動的にリンクされたライブラリ及びユーザレベルネーム空間を使用することができる。

【００５１】図５はこのようなユーザレベルバックアップファイルシステム５０１を示す。システム５０１は主システム５１１とバックアップシステム５１３の２つのコンピュータシステムで実現される。主システム５１１はアプリケーションプロセス５０３を実行し、バックアップシステム５１３はアプリケーションプロセス５０３により変更されたファイルのバックアップコピーを保持する。主システム５１１及びバックアップシステム５１３は、主システム５１１で実行するプロセスからのメッセージをバックアップシステム５１３で実行するプロセスに送信することができる通信媒体により接続されている。

【００５２】主システム５１１におけるシステム５０１のコンポーネントはアプリケーションプロセス５０３とカーネルサーバ３０５（ａ）である。カーネルサーバ３０５（ａ）は主システム５１１にファイルシステムを供給する。図５において、ファイルシステムは主システム５１１に対してローカルのディスク３０７（ａ）により示されるが、このファイルシステムは別のシステムと一緒に配置された遠隔ファイルシステムであることもできる。

【００５３】何れの場合も、カーネルサーバ３０５（ａ）はファイルシステムに対してファイル操作を行い、ファイルシステムはアプリケーションプロセス５０３からのコール３１７に応答し、そして、プロセス５０３へ結果３１９を返送し、また、ファイルシステム自体がディスク３０７（ａ）に対して必要な操作を行う。カーネルサーバ３０５（ａ）においてファイル操作を行うために、アプリケーションプロセス５０３は動的リンク可能なライブラリを使用する。

【００５４】主システム５１１では、このライブラリは、ｌｉｂ．３ｄ５０７と呼ばれる新たなライブラリにより置換される。ライブラリ５０７はコール３１７に応答し、適正なコール３１７をカーネルサーバ３０５に供給することによるだけでなく、バックアップメッセージ５１２をバックアップシステム５１３に送信することによっても特定のファイルを変更するファイル操作を指定する。変更によりバックアップメッセージ５１２を送信するファイルはフロントエンド複製ツリー（ＦＲＴ）５０５において指定される。

【００５５】フロントエンド複製ツリー（ＦＲＴ）５０５は、矢線５０６により示されるように、ｌｉｂ．３ｄ５０７におけるルーチンにより保持され、かつ、使用される。従って、複製ツリー５０５は、変更によりシステム５１３におけるバックアップファイルの変更を生じるファイルからなるユーザレベル論理ファイルシステムを定義する。

【００５６】バックアップシステム５１３におけるシステムのコンポーネントは、バックエンドサーバ５１５，ユーザレベルプロセス，カーネルサーバ３０５（ｂ）及びディスク３０７（ｂ），バックアップシステム５１３用の標準的なファイルシステムサーバ及びディスクドライブである。カーネルサーバ３０５（ｂ）はバックエンドサーバ５１７にファイルシステムを供給する。図５において、ファイルシステムに関するデータはローカルディスク３０７（ｂ）に記憶される。しかし、遠隔システムに記憶することもできる。

【００５７】バックエンドサーバ５１５は、コール３１７によりカーネルサーバ３０５（ｂ）に対してファイル操作を行い、そして、コールの結果をサーバ３０５（ｂ）から受信する。バックエンドサーバ５１５はバッ

クエンドマップ５１７を保持する。このバックエンドマップ５１７は、バックエンド複製ツリー５０５により指定されるファイルを、これらのバックアップとして役立つ、バックアップシステム５１３のファイルシステム内のファイルにマップする。カーネルサーバ３０５（ａ）により生成されたファイルシステム及びカーネルサーバ３０５（ｂ）により生成されたファイルシステムが同じネーム空間を有する実施例では、バックエンドマップ５１７は不要である。

【００５８】システム５０１の動作方法を図６に示す。図６は、ファイルを変更するライブラリ５０７におけるルーチン６０１の形態の概観図である。ルーチンが有するルーチンネーム６０３及び引数６０５は、ライブラリ５０７に置換されるライブラリにおいてファイル操作を行うのに使用されるファンクションのネーム及び引数と同一である。従って、アプリケーションプログラム５０９におけるこのルーチンの呼出はルーチン６０１を呼び出す。

【００５９】準備が必要なものを全て行った後、ルーチン６０１は、ルーチン６０１により置換されるルーチンと同じファイル操作をカーネルサーバ３０５（ａ）に行わせる。操作が成功すれば、ルーチン６１３は変更されるファイルのネームによりファンクション６１３を呼出し、フロントエンド複製ツリー５０５が、変更されるファイルがバックアップすべきものであることを示しているか否か決定する。フロントエンド複製ツリー５０５がそのように示している場合、ファンクション６１５は引数６１７を有するメッセージ５１２をバックアップシステム５１３に送る。

【００６０】このメッセージは、サーバ３０５（ａ）により供給されたファイルシステムで行われた操作と全く同じバックアップファイルシステムに対する操作をバックアップシステム５１３が行うことを要求する。メッセージを送った後、ルーチン６０１はリターンする。ファイルがフロントエンド複製ツリー５０５に存在しない場合又はファンクション６０７により指定される操作が成功しなかった場合も、ルーチン６０１はリターンする。

【００６１】図６において符号６１１が付されたコードのセクションは副作用（この場合はメッセージ５１２）を指定する。ルーチンの特徴は、主システム５１１でファイル操作が成功する場合だけメッセージ５１２をバックアップシステム５１３に送ることである。なぜなら、不成功操作はバックアップする必要がないからである。

【００６２】システム５０１におけるファイル操作には一般的に２つの形態がある。一つは、フロントエンド複製ツリー５０５及びバックエンドマップ５１７により実現されるユーザレベルネーム空間４０５変更することであり、もう一つは、変更しないことである。第２の形態の操作は例えば、フロントエンド複製ツリー５０５において指定されるファイルへの書込みである。ｌｉｂ．３ｄ５０７における書込ファンクションは、ｌｉｂ．３ｄに置換されるライブラリにおける書込ファンクションと同じインターフェースを有する。

【００６３】好ましい実施例では、このファンクションは引数として、ファイルを識別するためにカーネルサーバ３０５（ａ）により使用される整数のファイル記述子，被書込データを含有するバッファに対するポインタ及び被書込データのサイズを示す整数を有する。ｌｉｂ．３ｄにおける書込ファンクションは、カーネルサーバ３０５（ａ）がファイル記述子により指定されたファイルにシステム書込操作を行うことを要求し、操作が成功すれば、ファンクションはファイル記述子により識別されたファイルがフロントエンド複製ツリー５０５内に存在するか否かチェックし、存在すれば、ファンクションは書込メッセージ５１２をバックアップシステム５１３内のバックエンドサーバ５１５に送り、そして、リターンする。

【００６４】メッセージはカーネルサーバ３０５（ａ）により書込まれたファイルを識別し、そして、カーネルサーバ３０５（ａ）により供給されたファイルシステムにおけるシステム書込操作により行われたバックアップファイルシステムにおける書込操作を正確に行うのに必要な情報を包含する。バックエンドサーバ５１５がメッセージを受信すると、バックエンドサーバ５１５はバックエンドマップ５１７を使用し、カーネルサーバ３０５（ｂ）がバックアップファイル用に使用するファイル記述子を決定し、そして、その後、カーネルサーバ３０５（ｂ）により供給されたシステム書込ファンクションを使用し、メッセージ内に与えられたデータ及び位置情報を用いてバックアップファイルへの書込操作を実行する。

【００６５】ユーザレベルネーム空間４０５を変更する操作の単純なケースはファイル削除である。ｌｉｂ．３ｄにより供給される削除ファンクションは最初に、カーネルサーバ３０５（ａ）がファイルを削除することを要求し、この削除が行われると、削除ファンクションは、削除されたファイルに関する情報をフロントエンド複製ツリー５０５から削除する必要があるか否かチェックし、削除する必要がある場合、削除ファンクションはこの情報を削除する。

【００６６】次に、削除ファンクションは削除に必要なメッセージをバックエンドサーバ５１５に送り、そしてリターンする。バックエンドサーバ５１５がこのメッセージを受信すると、削除ファンクションはファイルをバックエンドマップ５１７に配置し、カーネルサーバ３０５（ｂ）がこのファイルを削除することを要求し、同時に、この削除により必要とされるバックエンドマップ５１７に対する全ての操作を実行する。

【００６７】一層複雑な事例はリネームである。カーネルサーバ３０５（ａ）により供給されるファイルシステ

ムにおけるファイルのリネームは、ユーザレベルネーム空間４０５に次の３つの可能性のある結果を有することができる。

【００６８】①ファイルの元のネームがユーザレベルネーム空間４０５の一部であり、新たなネームもユーザレベルネーム空間４０５の一部である場合、ファイルはユーザレベルネーム空間４０５内にとどまる。

【００６９】②ファイルの元のネームがユーザレベルネーム空間４０５の一部でなく、新たなネームもユーザレベルネーム空間４０５の一部でない場合、ファイルはユーザレベルネーム空間４０５に加えられる。

【００７０】③ファイルの元のネームがユーザレベルネーム空間４０５の一部であり、新たなネームがユーザレベルネーム空間４０５の一部でない場合、ファイルはユーザレベルネーム空間４０５から削除される。

【００７１】第１のケースでは、ｌｉｂ．３ｄのリネームファンクションは、カーネルサーバ３０５（ａ）がそのファイルシステムでリネームを行うことを要求する。次いで、リネームされたファイルがユーザレベルネーム空間４０５に存在するか否かチェックし、存在すれば、リネームファンクションはこのリネームを反映するためにフロントエンド複製ツリー５０５を変更し、メッセージをバックエンドサーバ５１５に送り、そこにおけるリネームを要求し、そしてリターンする。言うまでもなく、このメッセージは新旧のパスネームを包含する。バックエンドサーバ５１５がメッセージを受信すると、サーバ５１５はカーネルサーバ３０５（ｂ）のリネームを要求する。

【００７２】第２のケースでは、リネームファンクションはサーバ３０５（ａ）からリネームを要求し、そして前記のようにリネームされたファイルがユーザレベルネーム空間４０５内に存在するか否かチェックする。しかし、この場合、リネームファンクションはフロントエンド複製ツリー５０５からリネームファイルの情報を削除し、メッセージをバックエンドサーバ５１５に送り、リターンする。バックエンドサーバ５１５へのメッセージはファイルの削除メッセージである。このメッセージに応答して、バックエンドサーバ５１５は、カーネルサーバ３０５（ａ）にバックアップファイルを削除させる。

【００７３】第３のケースでは、リネームファンクションは再び前記のようなリネームを要求する。しかし、この場合、２種類のメッセージを送らなければならない。第１のメッセージは、ユーザレベルネーム空間４０５に移動されたファイルのネームを有するファイルをバックアップシステム５１３内に作成することを要求し、バックエンドサーバ５１５は、カーネルサーバ３０５（ｂ）がこのファイルを作成し、そして、その後、バックエンドマップ５１７にこのファイルのエントリを作成することを要求することによりこのメッセージに応答する。その後、リネームファンクションはユーザレベルネーム空間４０５に移動されたファイルの現在の内容を有する書込メッセージを送り、バックエンドサーバ５１５は、カーネルサーバ３０５（ｂ）にその内容をバックアップシステム５１３内のバックアップファイルに書き込ませることにより、この書込メッセージに応答する。

【００７４】前記の説明から明らかなように、主システム５１１におけるカーネルサーバ３０５（ａ）により行われる単一の操作は、バックエンドサーバ５１５がカーネルサーバ３０５（ｂ）に一連の操作を行わせることを要求する。更に前記の説明から明らかなように、ｌｉｂ．３ｄ５０７におけるファンクションにより行われる操作の終了時に、バックエンドマップ５１７とフロントエンド複製ツリー５０５は常に同じ状態である。

【００７５】好ましい実施例の実現：図７～１１

図７はユーザレベルバックアップファイルシステムの好ましい実施例７０１のブロック図である。この好ましい実施例は、或るプロセッサがＵＮＩＸ（登録商標）オペレーティングシステムのＳｕｎＯＳ４．１バージョンを実行し、他のプロセッサがＵＮＩＸ（登録商標）オペレーティングシステムのＭＩＰＳ４．５バージョンを実行するシステムで実現される。システム７０１には２つのコンポーネント群が存在する。一方の群はバックアップファイル操作を行い、他方の群はシステム７０１をフォルトトレラント（誤り許容）システムにする。下記の記載では、バックアップファイル操作を行うコンポーネントについて先ず説明し、次いで、フォルトトレランスを与えるコンポーネントについて説明する。

【００７６】主システム５１１から説明を始めると、アプリケーションプロセス５０３はアプリケーションプログラム５０９，動的にリンク可能なライブラリｌｉｂ．３ｄ５０７（この機能はファイル操作の副作用としてバックアップファイル操作を行う）及びフロントエンド複製ツリー（ＦＲＴ）５０５を有する。ファイル操作はシステム５１１によりカーネルサーバ３０５（ａ）により行われる。ライブラリ５０７におけるファンクションにより生成されるメッセージはパイプ７１０によりバックアップシステム５１３に搬送される。

【００７７】パイプ７１０はパイププロセス７１１によりアプリケーションプロセス５０３に供給され、パイププロセス７１１自体はパイプ７０９によりアプリケーションプロセス５０３と通信する。下記で更に詳細に説明するように、パイププロセス７１１は単一のパイプ７１０を供給する。パイプ７１０はバックアップシステム５１３でバックアップを作成する全てのアプリケーションプロセス５０３により使用される。

【００７８】好ましい実施例においてバックアップシステム５１３を続ける場合、バックエンドサーバ５１５は２つのプロセス、すなわち、バックエンドログプロセス（ＢＬＰ）７１６とシステム呼出エンジン（ＳＹＳＣＡＬＬ　ＥＮＧ）７１５に分割される。両方ともファイル

操作を行うためにカーネルサーバ３０５（ｂ）を使用する。バックアップファイルの他に、カーネルサーバ３０５（ｂ）により維持されるファイルシステムはログファイル７０３（ｂ）を包含する。

【００７９】操作は次の通りである。アプリケーションプロセス５０３が初期化される場合、パイププロセス７１１からパイプ７１０を指定するファイル識別子を取得するアプリケーションプログラム５０９の実行によりファイル操作の性能が生じる場合、ｌｉｂ．３ｄ５０７における操作のファンクションは、カーネルサーバ３０５（ａ）により供給されるファイルシステムに対するファンクションをカーネルサーバ３０５（ａ）に行わせ、更に、パイプ７１０を介してメッセージをバックアップシステム５１３に送信する。

【００８０】メッセージがバックアップシステム５１３に達すると、メッセージはバックエンドログプロセス（ＢＬＰ）７１６により受信される。バックエンドログプロセス７１６はカーネルサーバ３０５（ｂ）により供給されるファイルシステムにおけるログファイル７０３（ｂ）にメッセージをログする。ログファイル７０３（ｂ）がファイル内にメッセージを有する場合は常に、メッセージは到着順にシステム呼出エンジン７１５により読み出される。

【００８１】好ましい実施例では、バックエンドマップ５１７はシステム呼出エンジン７１５に属する。システム呼出エンジン７１５がメッセージを読み出すのに応じて、システム呼出エンジン７１５はカーネルサーバ３０５（ｂ）に、メッセージにより求められたファイル操作を行わせ、そしてシステム呼出エンジン７１５自体はメッセージにより求められるに応じてバックエンドマップ５１７を保持する。

【００８２】システム７０１の誤り許容操作

システムの誤り許容操作は、誤りが検出されること及び検出された誤りがシステムが操作を続行できるような方法で応答されることを必要とする。好ましい実施例では、誤りの検出及びこの検出に対する応答はＷａｔｃｈＤ（分散システム誤り許容を作成するユーザレベルシステム）により処理される。

【００８３】ＷａｔｃｈＤは、１９９３年６月２２日～２４日にフランス国のツールーズで開催された第２３回フォルト・トレラント・コンピューティングに関する国際会議における、ヒューアン，ワイ(Huang,Y)及びキンターラ，シー(Kintala,C)の“フォルトトレランスが実現されたソフトウエア：技術と経験(Software Implemented Fault Tolerance: Technologies and Experiences)”及び１９９２年９月３０日に出願されたヒューアン，ワイ(Huang,Y)の米国特許出願第０７／９５４，５４９号明細書（発明の名称：フォルトトレラントコンピューティングのための装置及び方法）に詳細に説明されている。

【００８４】本発明の説明では、ＷａｔｃｈＤシステムは、ｌｉｂｆｔと呼ばれるライブラリと分散システムの各ノードに対する一つのモニタプロセスを包含することを理解するだけでよい。ｌｉｂｆｔは、ＷａｔｃｈＤにプロセスを登録するような操作を行うルーチン，自動バックアップのためのメモリの領域を指定するルーチン及びこれらメモリの領域の対するチェックポイント操作を行うルーチンを包含する。モニタは、ＷａｔｃｈＤ及びモニタ相互にも登録されるモニタユーザプロセスを処理する。

【００８５】モニタに登録されたプロセスに障害が発生したことをモニタが検出すると、モニタはそのプロセスを再始動する。ｌｉｂｆｔファンクションによりプロセスが再始動される場合、プロセスは何が起こったのかを決定する。分散システムの或るノードに対するユーザプロセスをモニタする過程で、モニタは重要なデータ（これもｌｉｂｆｔファンクションを用いることにより決定される）のコピーを分散システムの別のノードに移動する。

【００８６】モニタのノードに障害が発生する場合、他方のノードのモニタがこの障害を検出し、重要データの現行コピーを用いて他方のノードのユーザプロセスを再始動する。障害ノードが復旧される場合、そのモニタは他方のノードからの重要情報を用いてユーザプロセスを再始動する。そして、ユーザプロセスが再始動されたことを示すメッセージを送信する。他方のノードのモニタがこのメッセージを受信すると、他方のノードはそのノードで実行するユーザプロセスを終了する。

【００８７】一般的に、ＷａｔｃｈＤモニタは環状に配列され、各モニタは環内のその隣のモニタをモニタする。環内のノードの数及びユーザプロセスの重要データのコピーを受信するモニタの数は、ＷａｔｃｈＤに登録されたユーザプロセスが再始動出来なくなる前に分散システムのノードを何個障害させなければならないかを決定する。

【００８８】好ましい実施例では、主システム５１１及びバックアップシステム５１３はそれぞれＷａｔｃｈＤモニタを有する。モニタとシステム７０１のコンポーネントとの間の関係は点線７２１により示される。主システム５１１用のモニタはモニタ７１７である。点線７２１で示されるように、モニタ７１７はパイププロセス７１１，フロントエンドログプロセス７０５及びシステム５１３内のモニタ７１９を監視する。モニタ７１９はモニタ７１７，システム呼出エンジン７１５及びバックエンドログプロセス７１６を監視する。

【００８９】図７に示されるように、システム７１７はフロントエンドログプロセス７０５，パイププロセス７１１，システム呼出エンジン７１５，バックエンドログプロセス７１６における障害及びシステム５１３の障害を取り扱うことができる。フォルトトレランス（誤り許

容性）を与えるシステム７０１の部分を２つ有するデザインは次の２つの主要な目的を有する。

【００９０】①性能に関して、障害回復のオーバーヘッドが確実に小さいこと，

②障害及び障害回復がアプリケーションに対して確実に透過的であり、実行アプリケーションが絶対に停止されないこと。

【００９１】障害回復方法は、ＷａｔｃｈＤがシステム内で最も信頼できるコンポーネントであるという仮定に基づく。これは、ＷａｔｃｈＤが極めて簡単なタスクを行い、障害発生後に自己回復できるためである。

【００９２】次に、バックアップシステム５１３の障害からの回復を詳細に説明する。他のプロセスの障害からの回復も概観する。バックアップシステム５１３の障害から説明を始めると、このようなケースでは、システム７０１は次のように動作する。モニタ７１７がシステム５１３の障害を検出すると、モニタ７１７はパイププロセス７１１に知らせ、パイププロセス７１１はフロントエンドログプロセス７０５を作成し、パイプ７１０のファイル記述子をフロントエンドログプロセス７０５に対するパイプ７０７のファイル記述子に置き換える。

【００９３】アプリケーションプロセス５０３により使用されるメッセージファンクションがパイプ７１０の障害を検出すると、このメッセージファンクションはパイププロセス７１１からパイプ７１０用の新たなファイル記述子を要求する。パイププロセス７１１はメッセージファンクションに、フロントエンドログプロセス７０５に結合されたパイプ７０７のファイル記述子を与え、メッセージファンクションにより送信されたメッセージはバックエンドログプロセス７１６ではなくフロントエンドログプロセス７０５に行く。フロントエンドログプロセス７０５がこのメッセージを受信すると、フロントエンドログプロセス７０５はこのメッセージを主システム５１１内のログファイル７０３（ａ）に配置する。

【００９４】好ましい実施例では、メッセージファンクションはパイプ５１０の障害を次のように検出する。プロセス５０３はＴＣＰ－ＩＰプロトコルを使用し、パイプ７０１にメッセージを送信する。このプロトコルでは、先行メッセージが受信された場合だけ次のメッセージを送信することができる。従って、ライブラリルーチン５０７におけるファンクションにより使用されるメッセージファンクションは２種類のメッセージ、すなわち真正メッセージとダミーメッセージを送信することによりパイプ７１０にメッセージを送信する。メッセージファンクションがダミーメッセージを送信出来る場合、真正メッセージは到達する。システム５１３の障害が発生すると、パイプ７１０を介して送信されるメッセージは到達せず、ダミーメッセージも送信出来ない。

【００９５】バックアップシステム５１３が回復すると、モニタ７１９はシステム呼出エンジン７１５とバックエンドログプロセス７１６を再始動させ、その後、モニタ７１７に通知する。モニタ７１７はパイププロセス７１１に通知する。パイププロセス７１１はパイプ７１０のファイル記述子を取得し、そしてフロントエンドログプロセス７０５を終了させる。バックエンドログプロセス７１６がシステム５１３で再始動すると、プロセス７１６はカーネルサーバ３０５（ａ）からログファイル７０３（ａ）のコピーを取得し、そして、このコピーをログファイル７０３（ｂ）に付加する。その後、システム呼出エンジン７１５はログファイル７０３（ｂ）におけるメッセージの実行を再開する。

【００９６】ｌｉｂ．３ｄにより使用されるメッセージファンクションは、パイプ７０７に関するファイル記述子を取得する方法と同じ方法で、パイプ７１０に関するファイル記述子を取得する。次に、メッセージファンクションがメッセージを送信するためにパイプ７１０のファイル記述子を使用しようと試みると、この試みは失敗し、メッセージファンクションは再びパイププロセス７１１からパイプファイル記述子を要求する。今度は、メッセージファンクションはパイプ７１０のファイル記述子を受信し、バックエンドログファイルに再び接続される。

【００９７】残りの障害状態は次のように処理される。

①パイププロセス７２２の障害

モニタ７１７がこの障害を検出する。新たに再始動されたプロセスは、ＷａｔｃｈＤによりセーブされたプロセス状態からパイプ７１０への接続を検索する。その他のプロセスはこの障害・回復に気付かない。

【００９８】②システム呼出エンジン７１５の障害

モニタ７１９がこの障害を検出し、システム呼出エンジン７１５を再始動させる。ｌｉｂｆｔにより与えられるチェックポイント及び回復機能により、新たに再始動されたシステム呼出エンジン７１５は外部ファイルからその先行チェックポイント状態へ回復することができる。その他のプロセスはこの障害・回復に気付かない。

【００９９】③バックエンドログプロセス７１６の障害

モニタ７１９がこの障害を検出し、バックエンドログプロセス７１６を再始動させる。プロセス７１６はチェックポイントファイルからその状態を復元する。モニタ７１９は更に、バックエンドログプロセス７１６が再始動されたことをモニタ７１７に通知し、次いで、モニタ７１７はパイププロセス７１１に通知する。その後、プロセス７１１はパイプ７１０を新たなバックエンドログプロセス７１６に接続する。各アプリケーションファイル及びｌｉｂ．３ｄの次の書込はパイププロセス７１１から新たな接続を取得する。

【０１００】④フロントエンドログプロセス７０５の障害

フロントエンドログプロセス７０５の障害はシステム５１３の障害発生期間中のみ存在する。モニタ７１７がフ

ロントエンドログプロセス７０５の障害を検出すると、モニタはパイププロセス７１１に通知する。次いで、フロントエンドログプロセス７０５を再始動させ、そしてパイプ７０８をフロントエンドログプロセス７０５に再接続させる。アプリケーションプログラム５０９の次の書込は失敗し、その結果、ｌｉｂ．３ｄにおけるメッセージ送信ファンクションはパイププロセス７１１から新たなパイプ７０８に関するファイル記述子を取得する。

【０１０１】ユーザレベルネーム空間４０５の実現：図８～１１

全ての一連のファイルを、カーネルサーバ３０５（ａ）により供給されるファイルシステムからアプリケーションプロセス５０３へ指定するために、ユーザレベルネーム空間４０５を使用することができる。図８は、カーネルサーバ３０５（ａ）により供給されるファイルシステムのネーム空間８０１とユーザレベルバックアップファイルシステム７０１におけるユーザレベルネーム空間４０５との間の関係を示す。

【０１０２】ネーム空間８０１において、ファイルネームはツリー状に配列される。図８におけるツリーの葉（Ｂ，Ｄ，Ｅ，Ｇ，Ｉ，Ｍ，Ｎ）を形成するファイルはデータ又はプログラムを含有する。残りのファイルは他のファイルのリストである。このようなファイルはディレクトリと呼ばれる。ネーム空間８０１における全てのファイルは、ルート“／”で始まるパスネームによりカーネルサーバ３０５（ａ）に対して指定することができ、また、ルートからパスネームにより指定されるファイルのネームまで全てのファイルのネームを包含する。従って、ファイル“Ｄ”のパスネームは／Ａ／Ｃ／Ｄであり、ファイル“Ｌ”のパスネームは／Ｊ／Ｋ／Ｌである。

【０１０３】ユーザレベルバックアップファイルシステム７０１は、ファイルを包含するネーム空間８０１のサブツリーを指定することにより、バックアップすべきファイルを指定する。次いで、ファイルを変更するサブツリー内のファイルに対する操作はバックアップシステム５１３内のバックアップファイルに対して行われる。図８では、４個のサブツリー，８０３（ａ），８０３（ｂ）及び８０３（ｃ）がバックアップすべきものとして選択されている。

【０１０４】従って、ネーム空間８０１におけるデータファイルＤ，Ｅ，Ｇ，Ｉ，Ｍ又はＮに対する変更は、このデータファイルのバックアップファイルに対する変更を生じるであろう。同様に、ディレクトリＣ，Ｆ，Ｈ及びＬに対する変更はこれらのバックアップファイルに対する変更を生じるであろう。サブツリー内の全てのファイルがバックアップされるので、バックアップすべきファイルは、サブツリーのルートであるディレクトリのパスネームにより、ユーザレベルネーム空間４０５で指定される。例えば、サブツリー８０３（ａ）は、パスネーム／Ａ／Ｃ８０５（ａ）によりユーザレベルネーム空間４０５において指定される。

【０１０５】言うまでもなく、ユーザレベルネーム空間４０５はシステム呼出エンジン７１５により供給されるファイルシステムに対してもマップされなければならない。これはバックエンドマップ５１７において行われる。図９に示されるように、バックエンドマップ５１７はユーザレベルネーム空間４０５における各オープンファイルに関するエントリを包含する。エントリは、ユーザレベルネーム空間情報９０３とバックアップシステム情報９０５の５１３の２つの部分からなる。

【０１０６】ユーザレベルネーム空間情報９０３はユーザレベルネーム空間４０５におけるファイルを識別する。バックアップシステム情報９０５は、ユーザレベルネーム空間情報により識別されたファイルに対応する、カーネルサーバ３０５（ｂ）により供給されるシステム内のファイルを識別する。

【０１０７】バックエンドマップ５１７はネーム空間８０１のサブツリーを、カーネルサーバ３０５（ｂ）がバックエンドログプロセス７１６及びシステム呼出エンジン７１５に供給したファイルシステムのネーム空間９０７のサブツリーにマッピングすることを可能にする。このマッピングは、ネーム空間８０１のサブツリーのルートのパスネームをネーム空間９０７の対応するサブツリーのルートのパスネームにマッピングすることにより行われる。

【０１０８】ルートのパスネームはサブツリー内のファイルのパスネームのプレフィックスと呼ばれている。例えば、サブツリー８０３（ａ）におけるパスネームはプレフィックス／Ａ／Ｃを有する。サブツリー８０３（ａ）内のファイルＥのパスネームはＥである。ネーム空間９０７において、サブツリー９０９／Ｚは、ネーム空間８０１からプレフィックス／Ａ／Ｃをネーム空間９０７のプレフィックス／Ｚにマッピングすることにより、サブツリー８０３（ａ）に対応させる。マッピングが完了した後、ネーム空間８０１におけるパスネーム／Ａ／Ｃ／Ｅにより指定されるファイルの変更は、ネーム空間９０７におけるパスネーム／Ｚ／Ｅにより指定されるファイルの変更を生じる。

【０１０９】フロントエンド複製ツリー５０５の詳細：図１０

好ましい実施例では、ユーザレベルネーム空間４０５はフロントエンド複製ツリー５０５により実現される。図１０はフロントエンド複製ツリー５０５を詳細に示すブロック図である。フロントエンド複製ツリー５０５の２つの主要なコンポーネントはＲＴＲＥＥ１０１５とファイル記述子キャッシュ１０２７である。ＲＴＲＥＥ１０１５はバックアップされるべきファイルを有するサブツリー８０３のルートのパスネームがリンクされたリストである。

【０１１０】ファイル記述子キャッシュ１０２７は、ファイル記述子をデバイス及びエントリ（ｉｎｏｄｅ）識別子に関連させるアレーである。従って、この実現の形は、ＵＮＩＸオペレーティングシステムにより提供されたファイルシステムが３つの方法、すなわち、パスネームにより、整数ファイル記述子により及びファイルが駐在するディスクの識別子とＵＮＩＸファイルシステムテーブル内のファイルのエントリ（ｉｎｏｄｅ）によりファイルを識別するという事実の結果である。

【０１１１】ファイルのファイル記述子はファイルをオープンしたプロセスだけに、また、このプロセスがこのファイルをオープンしている間だけ有効である。ＵＮＩＸファイルシステムテーブルは、パスネームとデバイス及びｉｎｏｄｅ間並びにデバイス及びｉｎｏｄｅと現行ファイル記述子との間の翻訳を可能にするが、パスネームと現行ファイル記述子との間の直接翻訳は可能にしない。

【０１１２】引き続いて更に詳細に説明すれば、ｍａｘｔｒｙ１００３及びｉｎｉｔ１００５はフロントエンド複製ツリー５０５の初期化に使用される。ｍａｘｔｒｙ１００３は、初期化機能がギブアップする前に、パイプ７１０をバックアップシステム５１３にセットアップするために試行した回数を示す。ｉｎｉｔ１００５は、パイプがセットアップされたか否かを示す。ＲＰＲＯＰアレー１００９はフロントエンド複製ツリー５０５で行うことができる操作のネーム１０１１のアレーである。

【０１１３】ＲＴＲＥＥ ＰＴＲ１０１３は、ＲＴＲＥＥリスト１０１５の最初の要素のポインタ、すなわち、各複製ツリー８０３の或る要素を包含するリンクリストである。各要素１０１７は複製ツリー８０３のルートのパスネーム１０２１，パスネーム１０２１の長さ１０１９及びリンクリスト内の次の要素に対するポインタ１０２３を包含する。接続サーバ１０２５はバックアップシステム５１３に対するパイプ７１０のネーム空間８０１におけるパスネームである。

【０１１４】ＦＤキャッシュ１０２７はファイル記述子キャッシュエントリ１０２９のアレーである。このアレー内のエントリ１０２９は、アプリケーションプロセス５０３に対して利用可能なファイル記述子と同じくらい多数存在する。ＦＤキャッシュ１０２７内の所定のファイル記述子のエントリのインデックスはファイル記述子である。エントリ１０２９は、エントリが現に有効であるか否かを示し、かつ、ファイルがオープンされている間、アプリケーションプロセス５０３が子を産むか否かを示す状態フラグを包含する。

【０１１５】エントリ１０２９は主システム５１１におけるファイル駐在デバイスの識別子１１０１と主システム５１１におけるファイルのｉｎｏｄｅの識別子も包含する。ＲＴＲＥＥ１０１５におけるエントリにより指定されるサブツリー８０３で現にオープンされている各ファイルの有効エントリ１０２９が存在する。

【０１１６】バックエンドマップ５１７の詳細

バックエンドマップ５１７は、パスネームマップ１１１３とオープン複製ファイルリスト１１１７の２つの部分からなる。パスネームマップ１１１３は、主システム５１１のネーム空間８０１におけるパスネームをバックアップシステム５１３のネーム空間９０７のパスネームにマップするだけである。マップ内の各エントリ１１１５は、フロントエンドパスネーム１１１７とバックエンドパスネーム１１１９との間の関係を確立する。

【０１１７】パスネームマップ１１１３には、フロントエンドネーム空間９０７におけるサブツリー８０３のルートをネーム空間９０７におけるサブツリーのルートにマッピングするエントリが包含されている。バックエンドパスネーム１１１９はバックアップシステム情報９０５の一部である。好ましい実施例では、これらのマッピングはシステム構成ファイルにおいて指定される。

【０１１８】オープン複製ファイルリスト１１１７は、アプリケーションプロセス５０３がその複製ツリー８０３において現にオープンしている各ファイルのエントリ１１１９を包含する。エントリ１１１９におけるユーザレベルネーム空間情報９０３はフロントエンドファイル識別子（ＦＦＩＤ）１１０５及びフロントエンドパスネーム（ＦＰ）１１０６を包含する。フロントエンドファイル識別子（ＦＦＩＤ）１１０５は主システム５１１におけるファイルのデバイス識別子とｉｎｏｄｅ識別子から構成されている。

【０１１９】フロントエンドパスネーム（ＦＰ）１１０６はフロントエンドプレフィックス（ＦＰＲ）１１０７とサブツリーパスネーム１１０８に分割される。フロントエンドプレフィックス（ＦＰＲ）１１０７はフロントエンドネーム空間８０１におけるファイルのサブツリーのプレフィックスである。サブツリーパスネーム１１０８はそのサブツリー内のファイルのパスネームである。エントリ１１１７におけるバックアップシステム情報９０５はバックエンドファイル記述子１１１１からなる。

【０１２０】バックエンドファイル記述子１１１１は、このファイルについてカーネルサーバ３０５（ｂ）により供給されるファイルシステムにおけるファイル記述子である。好ましい実施例では、バックエンドマップ５１７はハッシュテーブルとして実現される。このハッシュテーブルは、フロントエンドファイル識別子１１０５及びフロントエンドパスネーム１１０６の両方によりアクセスできる。

【０１２１】データ構造５０５及び５１７を含む操作

データ構造５０５及び５１７を作成する方法及びこれらのデータ構造を様々なファイル操作により作用させる方法について説明する。好ましい実施例では、アプリケーションプロセス５０３はコーンシェルを使用するＵＮＩＸオペレーティングシステムで実行する。

【０１２２】コーンシェルは、プロセスがＥＮＶ変数を設定できるようにする。ＥＮＶ変数は、プロセスがコーンシェルを呼び出すときはいつでも、実行されるファイルを指定する。アプリケーションプロセス５０３においてＥＮＶ変数により指定されるファイルは、フロントエンド複製テーブル５０５を構築し、かつ、初期化するために、アプリケーションプロセス５０３にとって必要な情報を含有する。

【０１２３】作成されると、テーブル５０５は、アプリケーションプロセス５０３のアドレス空間の一部となり、アプリケーションプロセス５０３の子プロセスに対して利用可能である。この子プロセスは、ＵＮＩＸオペレーティングシステムのフォークシステム呼出により作成される。従って、子プロセスはその親の環境を受け継ぐ。一方、ｅｘｅｃシステム呼出は子プロセスに新たな環境を与える。

【０１２４】ｅｘｅｃシステム呼出により作成されたアプリケーションプロセス５０３の子に対して利用可能なフロントエンド複製ツリー５０５を作成するために、ｌｉｂ．３ｄは、新たなプロセスについて利用可能なＥＮＶにフロントエンド複製ツリー５０５をコピーするｅｘｅｃファンクションを有する。このため、プロセスが他の点で、その親のアドレス空間を受け継いでいない場合であっても、ｌｉｂ．３ｄはこのプロセスに対して利用可能である。その他の実施例は、フロントエンド複製ツリー５０５をｅｘｅｃにより作成された子プロセスにパスするための、ネーム付パイプ又は外部ファイルさえも使用できる。

【０１２５】ファイル操作について更に説明する。これらの操作のうちの最初の操作はマウント操作である。ＵＮＩＸオペレーティングシステムでは、マウントはネームのツリーをオペレーティングシステムのネーム空間に追加する。好ましい実施例では、ｌｉｂ．３ｄで実現されるマウントのバージョンは、フロントエンドネーム空間８０１のサブツリーを複製ツリー８０５としてユーザレベルネーム空間４０５に追加させるモードを包含する。

【０１２６】マウントがこのモードで使用される場合、パスネーム引数はユーザレベルネーム空間４０５に追加されるサブツリー８０３のルートのパスネームである。このファンクションは、このパスネームについて複製ツリーエントリ１０１７を作成し、そして、このエントリを複製ツリー１０１５に追加することにより、サブツリー８０３をユーザレベルネーム空間４０５に追加する。また、指定されたパスネームを有する複製ツリーエントリ１０１７を複製ツリー１０１５から削除するアンマウント操作も存在する。

【０１２７】アプリケーションプロセス５０３が複製ツリー８０５内のファイルに対してオープン操作を行う場合、ｌｉｂ．３ｄ内のオープンファンクションは、新たにオープンされたファイルに関するファイル記述子キャッシュエントリ１０２９を作成し、オープンメッセージをバックエンドログプロセス７１６に送信する。このオープンメッセージはオープンされたファイルの主システム５１１におけるパスネーム，デバイス識別子及びｉｎｏｄｅ識別子を含む。

【０１２８】このメッセージがシステム呼出エンジン７１５により実行されると、バックエンドマップ５１７でエントリ９０１が作成される。パスネーム１１３は、主システム５１１でオープンされているファイルに対応するバックエンドシステム５１３におけるファイルを発見するために使用される。対応するファイルのファイル記述子はバックエンドファイル記述子１１１１に配置される。

【０１２９】ファイルがオープンされると、主システム５１１におけるファイル操作は、ファイルを識別するためにファイル識別子を使用する。バックアップシステム５１３におけるバックアップファイルに対する対応する操作のためのメッセージは、ファイルを識別するためにデバイス識別子とｉｎｏｄｅ識別子を使用する。このようなメッセージを実行するために、システム呼出エンジン７１５は、メッセージ内で指定されたデバイス及びｉｎｏｄｅについて、オープン複製ファイルリスト１１１７内のエントリにアクセスするだけでよい。このエントリはバックアップシステム５１３を実行するのに必要なファイル記述子１１１１を包含する。

【０１３０】アプリケーションプロセス５０３が複製ツリー５０５内のファイルをクローズする場合、ｌｉｂ．３ｄクローズファンクションは、子プロセスがファイルを使用しているか否かについて、状態フィールド１０３３から決定する。子プロセスがファイルを使用していない場合、クローズファンクションは複製ツリー５０５内のファイルに関するファイル記述子キャッシュエントリ１０２９を無効にし、デバイス識別子及びｉｎｏｄｅ識別子を含むクローズメッセージをバックアップシステム５１３に送信する。

【０１３１】システム呼出エンジン７１５がこのメッセージを実行する場合、このファイルのエントリ１１１９を見つけるために、デバイス及びｉｎｏｄｅ識別子を使用する。次いで、ファイルを識別するためのバックエンドファイル記述子１１１１を使用しバックアップシステム５１３内のファイルをクローズし、そして、最後に、オープン複製ファイルリスト１１１７からエントリ１１１９を削除する。

【０１３２】以上、ユーザレベルバックアップファイルシステムの形成方法及び使用方法の一例について詳細に説明してきたが、本発明のユーザレベルバックアップファイルシステムの様々な変更例を実現できることは当業者に明らかである。例えば、好ましい実施例は、ＵＮＩＸオペレーティングシステムを実行するシステムで実現

され、好ましい実施例の多数の細部はＵＮＩＸオペレーティングシステムを実行するシステムで実現されることを反映する。その他のオペレーティングシステムおいて実現する場合、これらの細部は変更される。

【０１３３】同様に、好ましい実施例は、Ｓｕｎのオペレーティングシステムで使用される共用ライブラリシステムを使用する。その他の実施例は動的にリンクされたライブラリのその他の構成を使用する。更に、この明細書で開示したデータ構造及びメッセージは、この明細書に開示されたものと機能的に同等のその他の多くの方法により実現させることもできる。

【０１３４】更に、ユーザレベルバックアップファイルシステムは、プロセスにより実行されているアプリケーションプログラムを変更することなしにプロセスの挙動を変更するための、動的にリンクされた代替ライブラリの使用方法の一例にしか過ぎない。この同じ手法は、オペレーティングシステムによりプロセスに与えられたネーム空間と異なる、プロセス用ユーザレベルネーム空間を定義するためにも使用でき、また、ファイルバックアップばかりでなく、その他のサービスも、ユーザレベルネーム空間又はオペレーティングシステムがプロセスに与えるネーム空間の何れかのエンティティのネームに付加するためにも使用できる。

【０１３５】

【発明の効果】以上説明したように、本発明によれば、アプリケーションプログラム，オペレーティングシステム又はハードウエアを変更することなしに使用できるバックアップファイルシステムが得られる。

【図面の簡単な説明】

【図１】従来のバックアップファイルシステムのブロック図である。

【図２】ライブラリがユーザプログラムのインターフェースを再定義する方法を示す模式図である。

【図３】動的にリンクされたライブラリを用いてオペレーティングシステムインターフェースを再定義する方法を示すブロック図である。

【図４】動的にリンクされたライブラリを用いてユーザレベルネーム空間を供給する方法を示すブロック図である。

【図５】動的にリンクされたライブラリを使用するユーザレベルバックアップファイルシステムのブロック図である。

【図６】動的にリンクされたライブラリにおけるルーチンの概要図である。

【図７】ユーザレベルバックアップファイルシステムの好ましい実施例のブロック図である。

【図８】カーネルサーバ（ａ）により供給されるネーム空間とユーザレベルネーム空間との関係を示すブロック図である。

【図９】ユーザレベルネーム空間とカーネルサーバ（ｂ）により供給されるネーム空間との関係を示すブロック図である。

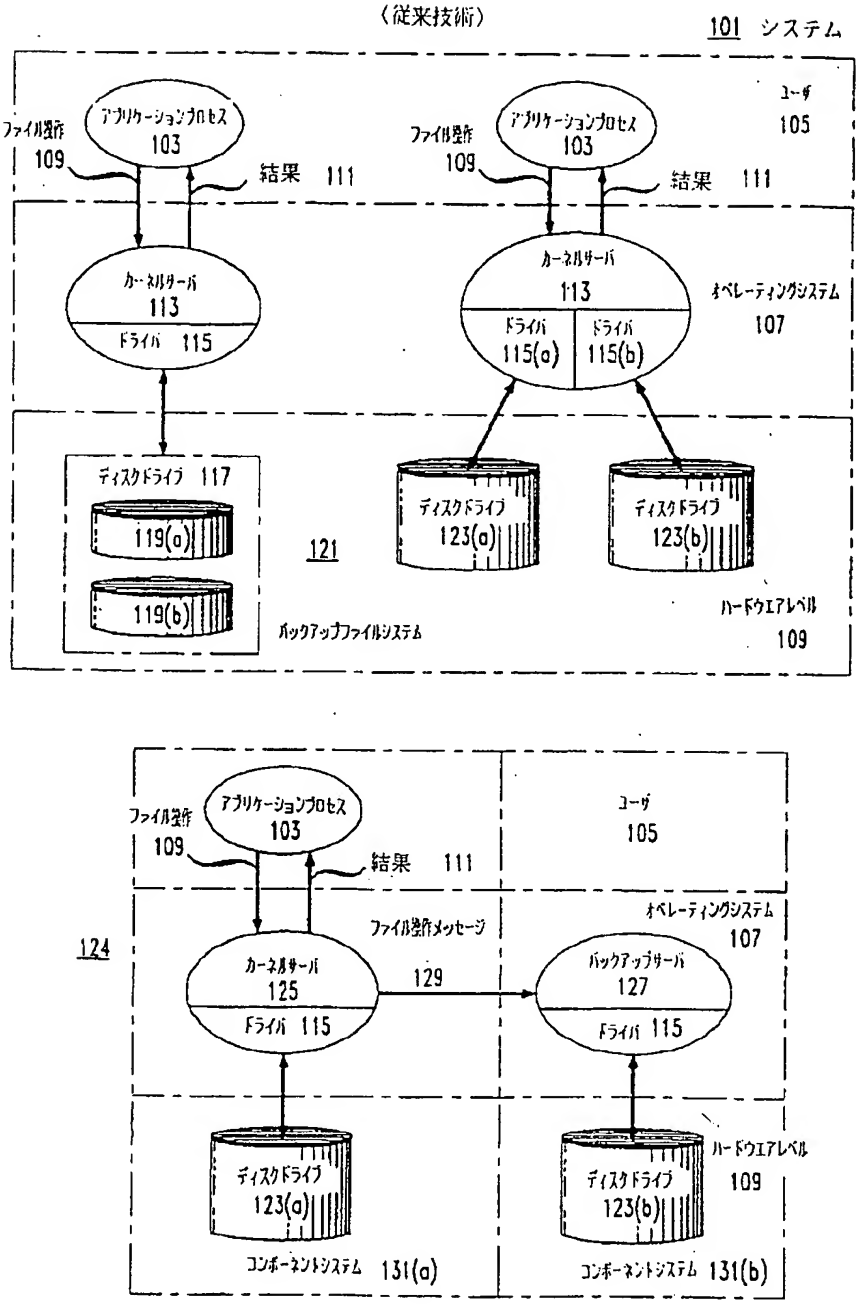
【図１０】フロントエンド複製ツリー５０５の構造を示すブロック図である。

【図１１】バックエンドマップ５１７の構造を示すブロック図である。

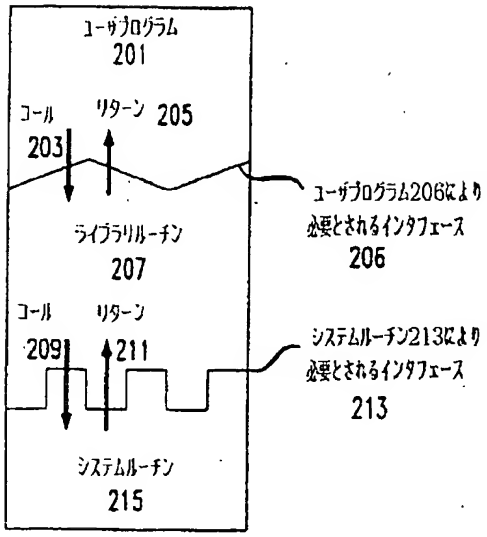
【符号の説明】

１０１　従来のバックアップファイルシステム
１０３　アプリケーションプロセス
１０５　ユーザ
１０７　オペレーティングシステム
１１３　カーネルサーバ
１１５　ドライバ
１１７　ディスクドライブ
３０５　カーネルサーバ
３０７　ユーザプロセス
３０９　アプリケーションプログラム
３１５，３２１　オペレーティングシステムライブラリ
４０３　オペレーティングシステムライブラリ
４０５　ユーザレベルネーム空間
４０９　ユーザプロセス
５０１　ユーザレベルバックアップファイルシステム
５０３　アプリケーションプロセス
５０５　フロントエンド複製ツリー
５０７　ｌｉｂ．３ｄ
５０９　アプリケーションプログラム
５１１　主システム
５１３　バックアップシステム
５１５　バックエンドマップ
５１７　バックエンドマップ
７０１　ユーザレベルバックアップファイルシステム
７０３　ログファイル
７０５　フロントエンドログプロセス
７０７，７０９，７１０　パイプ
７１１　パイププロセス
７１５　システム呼出エンジン
７１６　バックエンドログプロセス
７１７，７１９　モニタ
８０１　ファイルシステムネーム空間
９０３　ユーザレベルネーム空間情報（ＵＬＮＩ）
９０５　バックアップシステム情報（ＢＳＩ）
９０７　ネーム空間
９０９　サブツリー

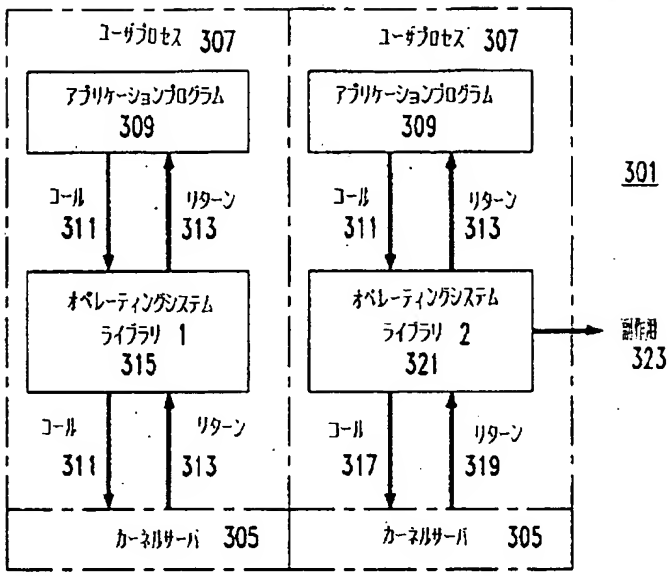
【図１】


（従来技術）
101 システム
ユーザ
105
アプリケーションプロセス
103
ファイル操作
109
結果 111
カーネルサーバ
113
ドライバ 115
ドライバ
115(a)
ドライバ
115(b)
オペレーティングシステム
107
ディスクドライブ 117
119(a)
119(b)
121
バックアップファイルシステム
ディスクドライブ
123(a)
ディスクドライブ
123(b)
ハードウエアレベル
109
124
ファイル操作メッセージ
バックアップサーバ
127
129
コンポーネントシステム 131(a)
コンポーネントシステム 131(b)

【図２】

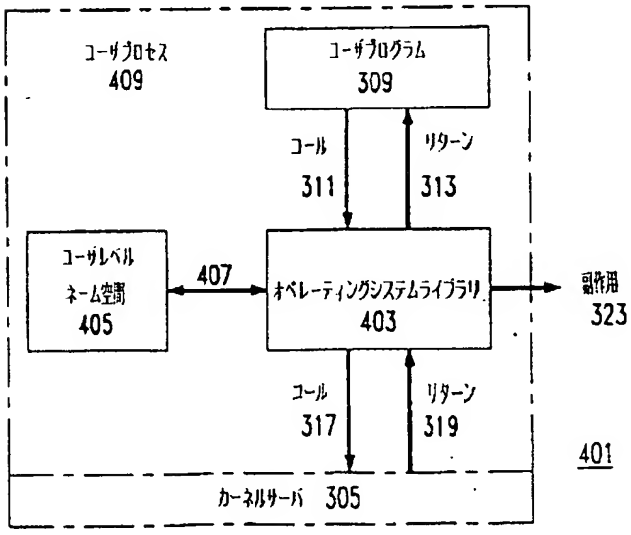

ユーザプログラム
201
コール
203
リターン
205
ユーザプログラム206により
必要とされるインタフェース
206
ライブラリルーチン
207
コール
209
リターン
211
システムルーチン213により
必要とされるインタフェース
213
システムルーチン
215

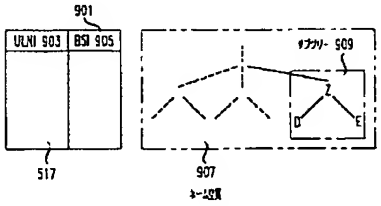
【図３】


ユーザプロセス 307
ユーザプロセス 307
アプリケーションプログラム
309
アプリケーションプログラム
309
301
コール
311
リターン
313
コール
311
リターン
313
オペレーティングシステム
ライブラリ 1
315
オペレーティングシステム
ライブラリ 2
321
副作用
323
コール
311
リターン
313
コール
317
リターン
319
カーネルサーバ 305
カーネルサーバ 305

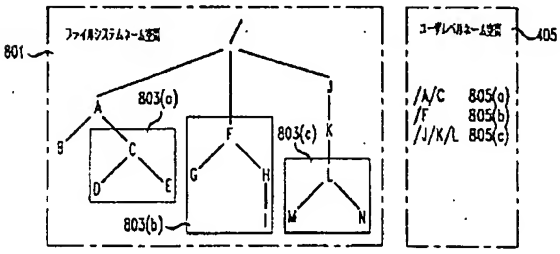
【図４】

【図９】

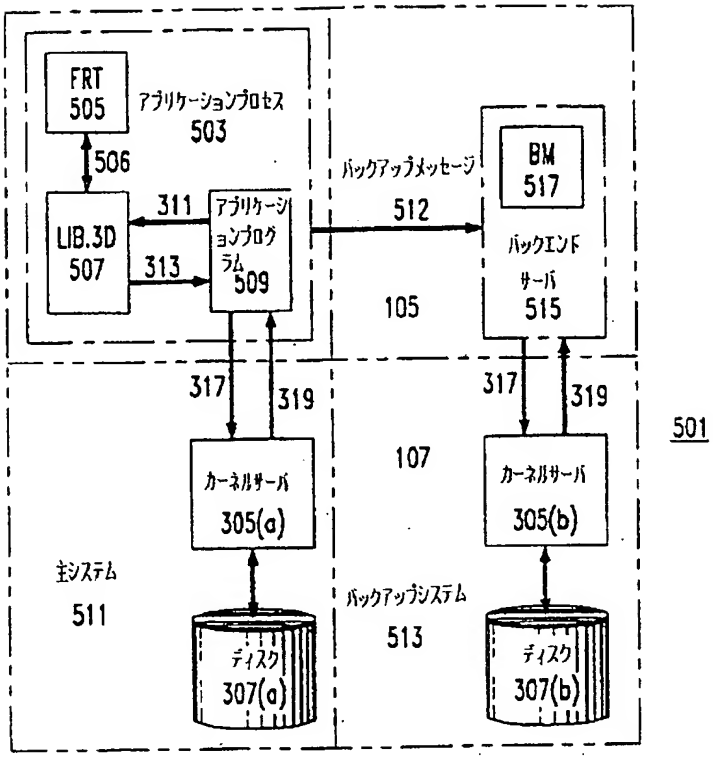

ユーザプロセス
409
ユーザプログラム
309
コール
311
リターン
313
ユーザレベル
ネーム空間
405
407
オペレーティングシステムライブラリ
403
副作用
323
コール
317
リターン
319
401
カーネルサーバ 305


901
ULN1 903
BSI 905
517
907

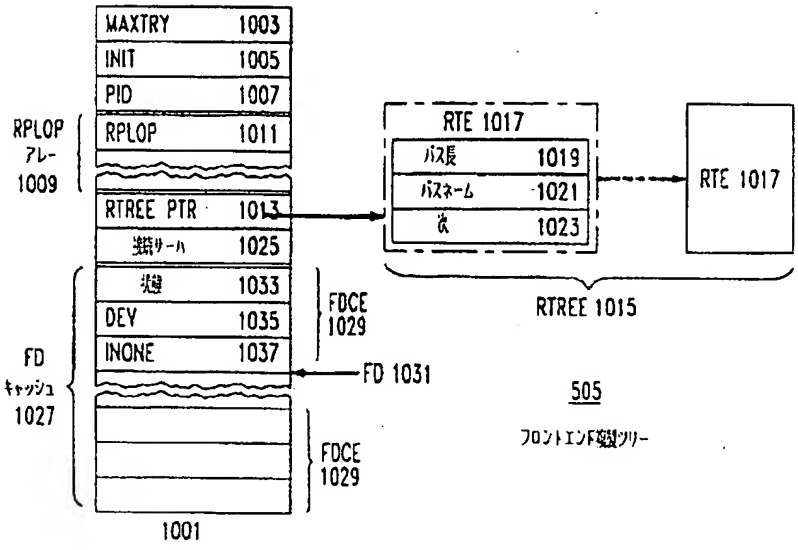
【図８】


801
803(a)
803(b)
803(c)
405
/A/C 805(a)
/F 805(b)
/I/K/L 805(c)

【図５】


FRT
505
アプリケーションプロセス
503
506
311
313
LIB.3D
507
アプリケーションプログラム
509
バックアップメッセージ
512
BM
517
バックエンドサーバ
515
105
317
319
317
319
501
107
カーネルサーバ
305(a)
カーネルサーバ
305(b)
主システム
511
バックアップシステム
513
ディスク
307(a)
ディスク
307(b)

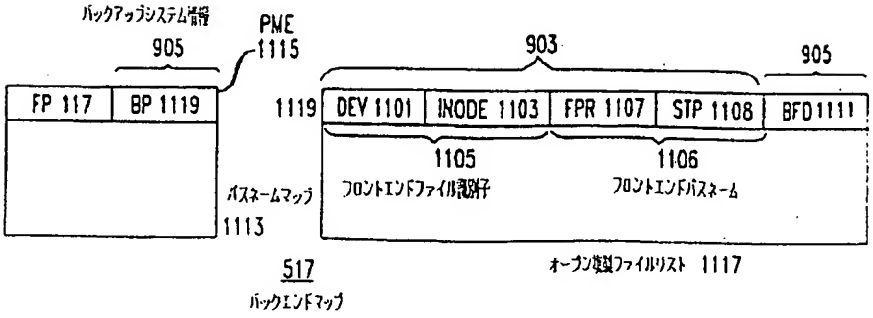
【図１０】


MAXTRY 1003
INIT 1005
PID 1007
RPLOP 1011
RPLOP アレー 1009
RTREE PTR 1013
1025
1033
DEV 1035
INONE 1037
FD キャッシュ 1027
FDCE 1029
FDCE 1029
FD 1031
1001
RTE 1017
パス長 1019
パスネーム 1021
1023
RTE 1017
RTREE 1015
505
フロントエンド変換ツリー

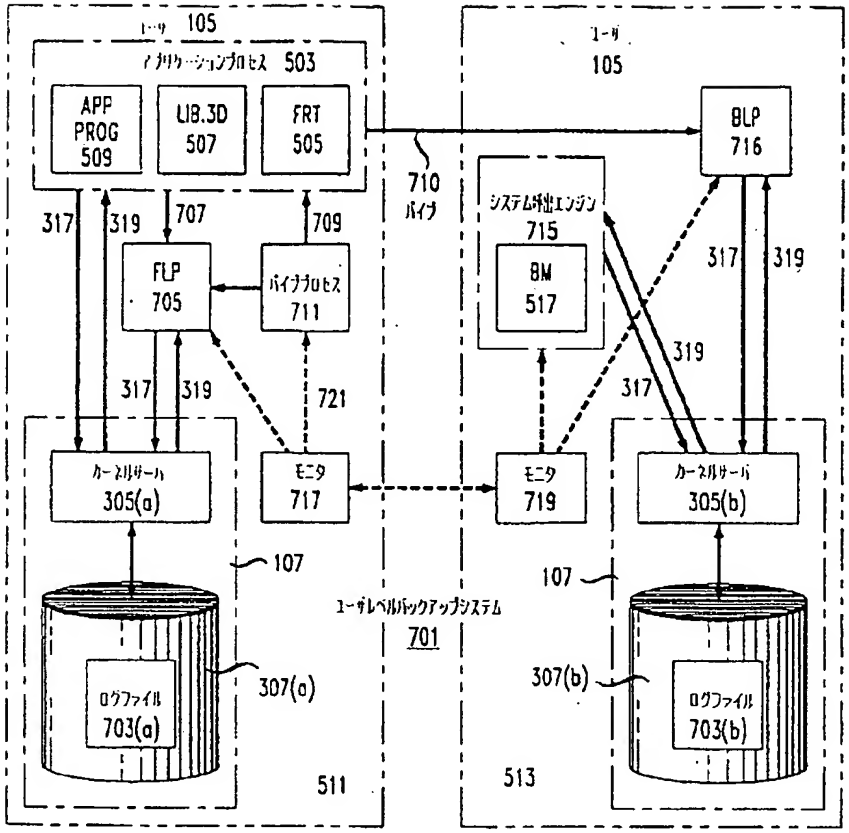
【図６】

```
603                                                    605
初期化<アプリケーションプログラム-ファイル操作>(<アプリケーションコード-ファイル操作-引数>)
{
    607                    609
  IF<サーバ-ファイル操作>(<サーバ-ファイル操作-引数>)
          613              614
    IF IN-領域ツリー(<ファイル-引数>)
{
            615          617
      { 送信-メッセージ(<メッセージ-引数)
        リターン(成功)
      }
    ELSE
        リターン(成功)
  ELSE
      リターン(成功)
}
```

【図１１】



【図７】



フロントページの続き

(72)発明者　エンナン　フアング
アメリカ合衆国、08807　ニュージャージー、ブリッジウォーター、リンバーガー　33

(72)発明者　デビッド　ジェラード　コーン
アメリカ合衆国、10003　ニューヨーク、ニューヨーク、エー　107、マーサー　ストリート　303

(72)発明者　チュング　ー　フワ　ハーマン　ラオ
アメリカ合衆国、08820　ニュージャージー、エジソン、スプリングブルック　ドライブ　4304